EXHIBIT A

Astute Content Processor Architecture

June 20, 2001

Astute Networks

Agenda

- Goals
- How does one NOT run TCP at 5-10 Gbps?
- Astute Architecture
- Individual Blocks
- System Aspects

Astute Networks

Goals

- 5-10 Gbps Full Duplex
  - Used in OC-192 networks.
- TCP Connection Rate ~ 500 kcps
  - Setup and teardown of a connection.
- Customer Level Programmability
  - Customer can add value and differentiate product.
- Acceleration of Customer's applications
  - Feeding a 10 Gbps FD byte stream to a Host CPU will overwhelm it. So how can we help?
- Scalability
  - Need an architecture that will scale with speed without "re-inventing the wheel".

How does one NOT run TCP @ 10 Gbps?

- Single CPU/NPU running faster
  - CPUs are not scaling as fast as the networking requirements.
  - Today, they can, at a stretch, handle 1 Gbps FD.
  - 2 Gbps FD in our timeframe.
  - Lots of bottlenecks in a general purpose CPU.
- Multiple General Purpose CPU/NPUs
  - Common architecture is multiple CPUs with "TCP software".
  - Bottleneck becomes contention for resources by the multiple CPUs.

Astute Networks

Astute Architectural Assumptions

- To run TCP at 10 Gbps, multiple CPUs will be required.
  - Even if we designed a single CPU that can run at 20 GHz, this will not scale.
  - It is better to have the risk in architecture rather than circuit layout.
- The CPUs cannot stall
  - Every time a CPU stalls, it's performance goes down.
  - This means they cannot clash while accessing shared resources.
- Use off-the-shelf embedded CPUs
  - Our value add is system design not CPU architecture
- Minimize external memory accesses
  - Let's not go overboard with the pinout

What do we do with the CPUs?

- Pass all the information required to the CPU so it can process it without stalling
  - The TCP State Information for the TCP Flow being processed
  - All the information in the incoming packet or message from the Host that the CPU may need – an Event
- Use multiple CPUs
- Balance the traffic between the CPUs dynamically
  - Handle any mix of TCP flows

Astute Networks

What else do we do?

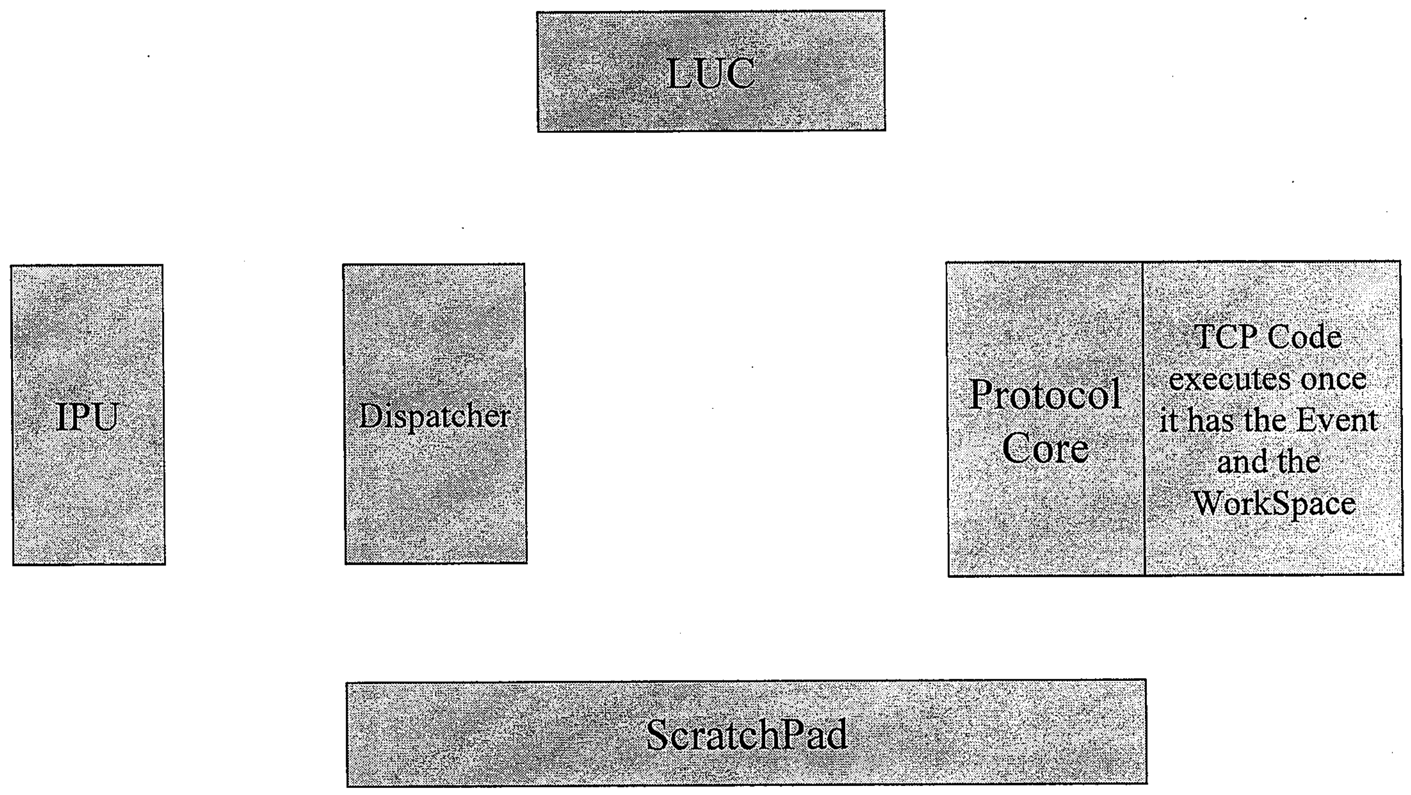
- Re-assemble the byte stream into memory
  - Do not store packets.
  - Hardware assisted data memory management
- Hardware Support for timers
  - 1M flows x 5 timers is a lot of timers to decrement on a regular basis.
- Internal ScratchPad
  - Store data temporarily while TCP decides what to do
  - Allows Customer software to examine packet contents
- Interface Cores
  - Embedded cores totally dedicated to customer value-add

Astute Networks

Astute Networks

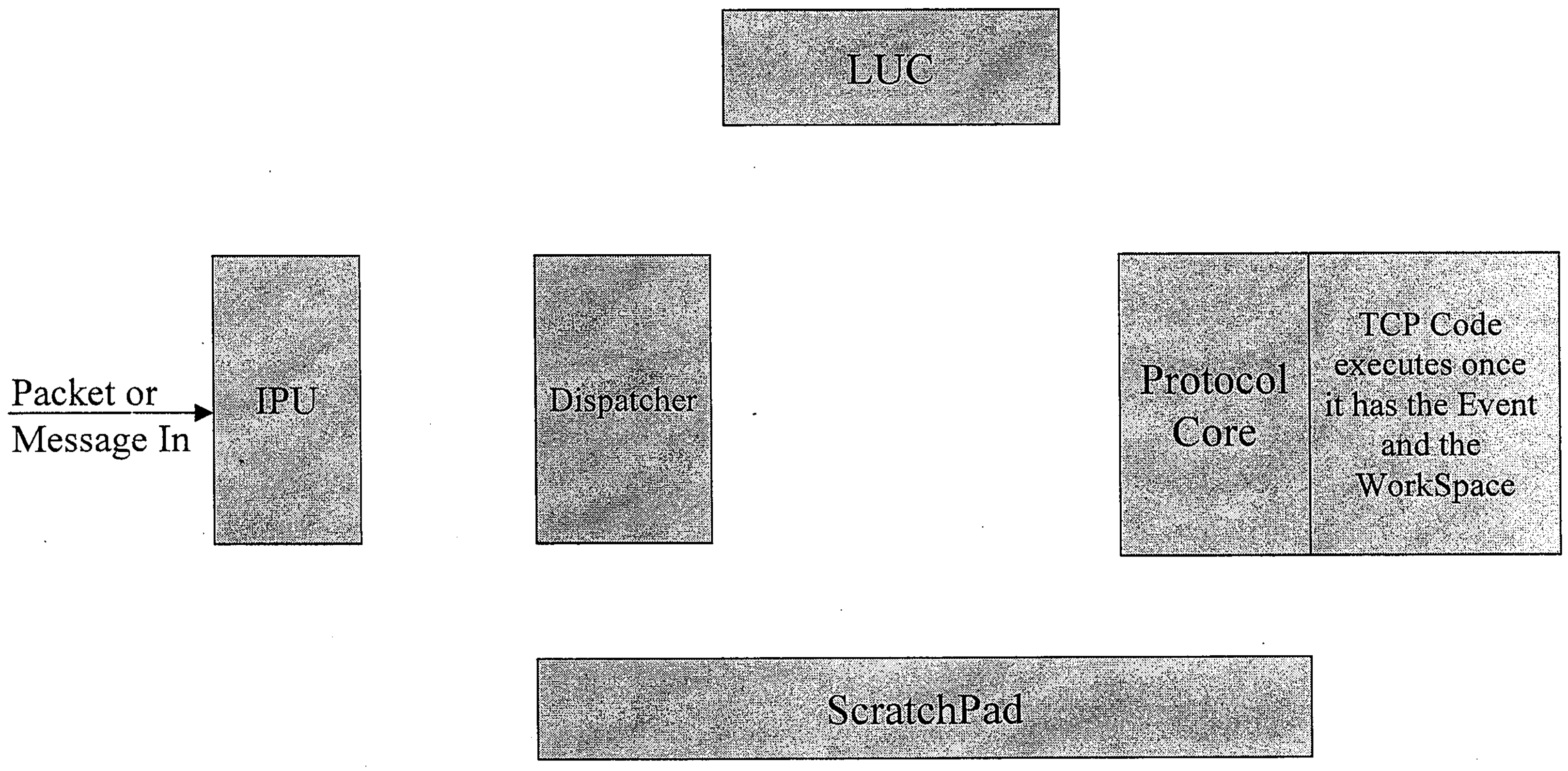
Pre-TCP DataPath

Pre-TCP DataPath



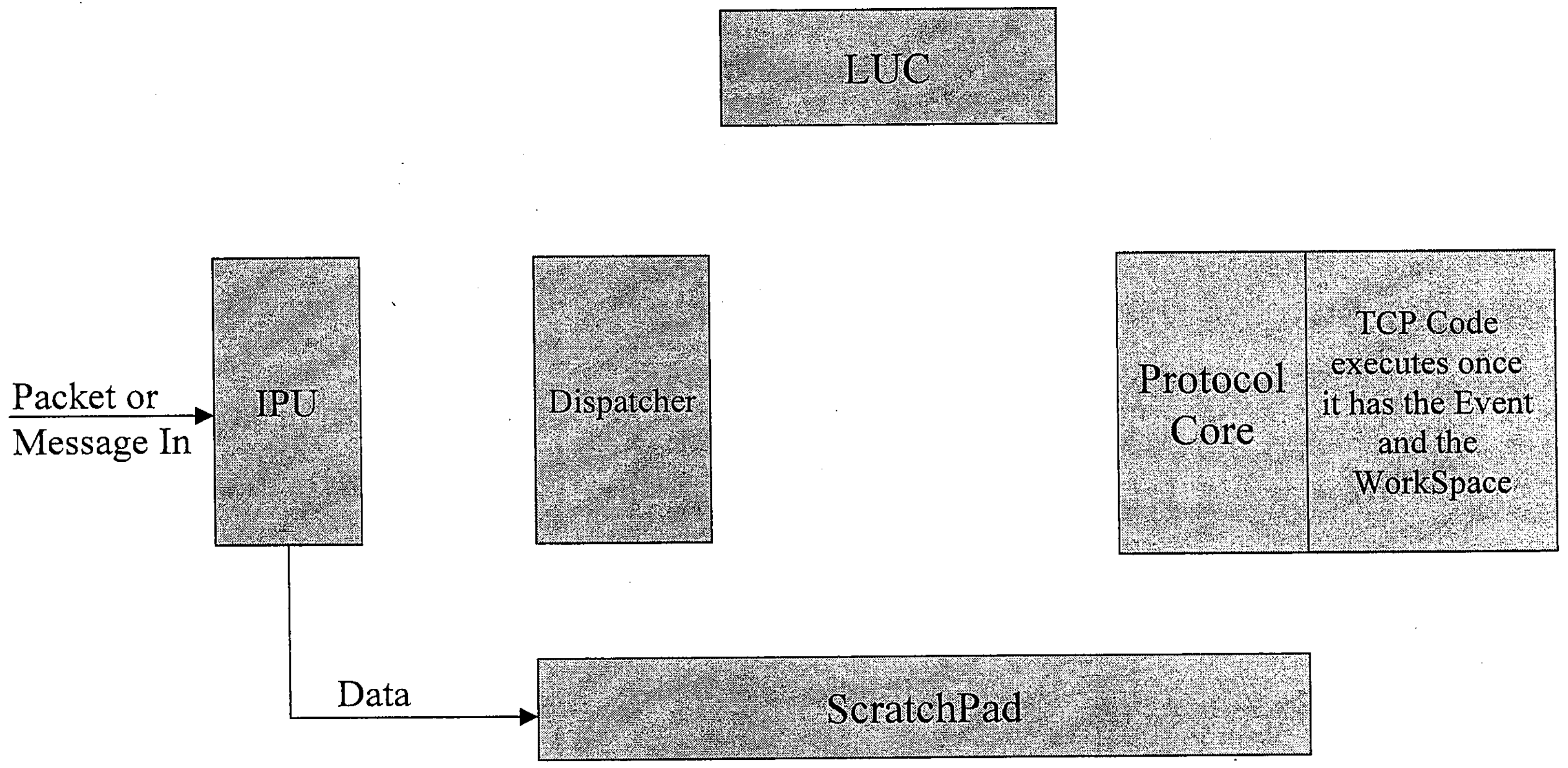
Astute Networks

Pre-TCP DataPath



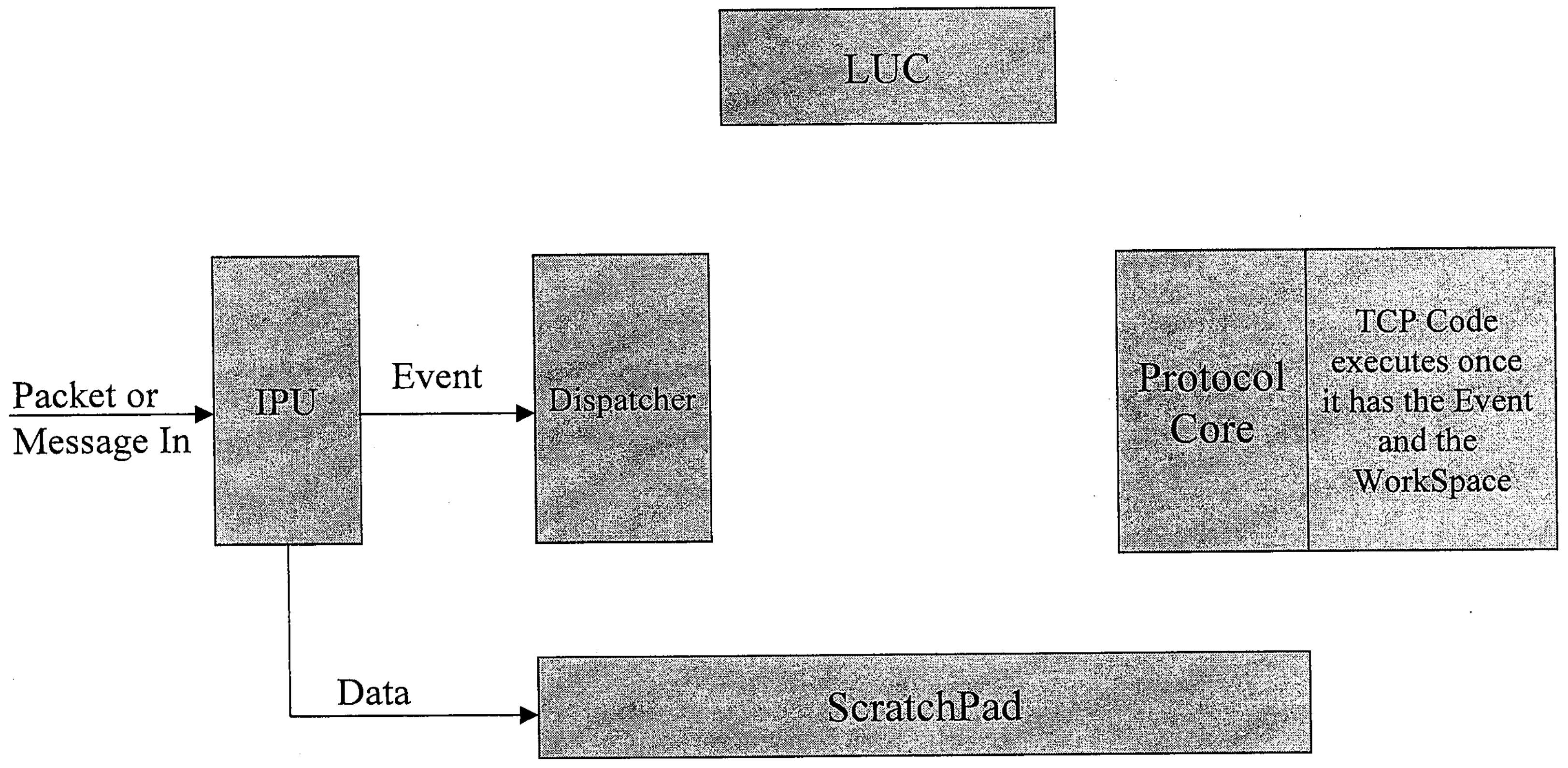
Astute Networks

Pre-TCP DataPath



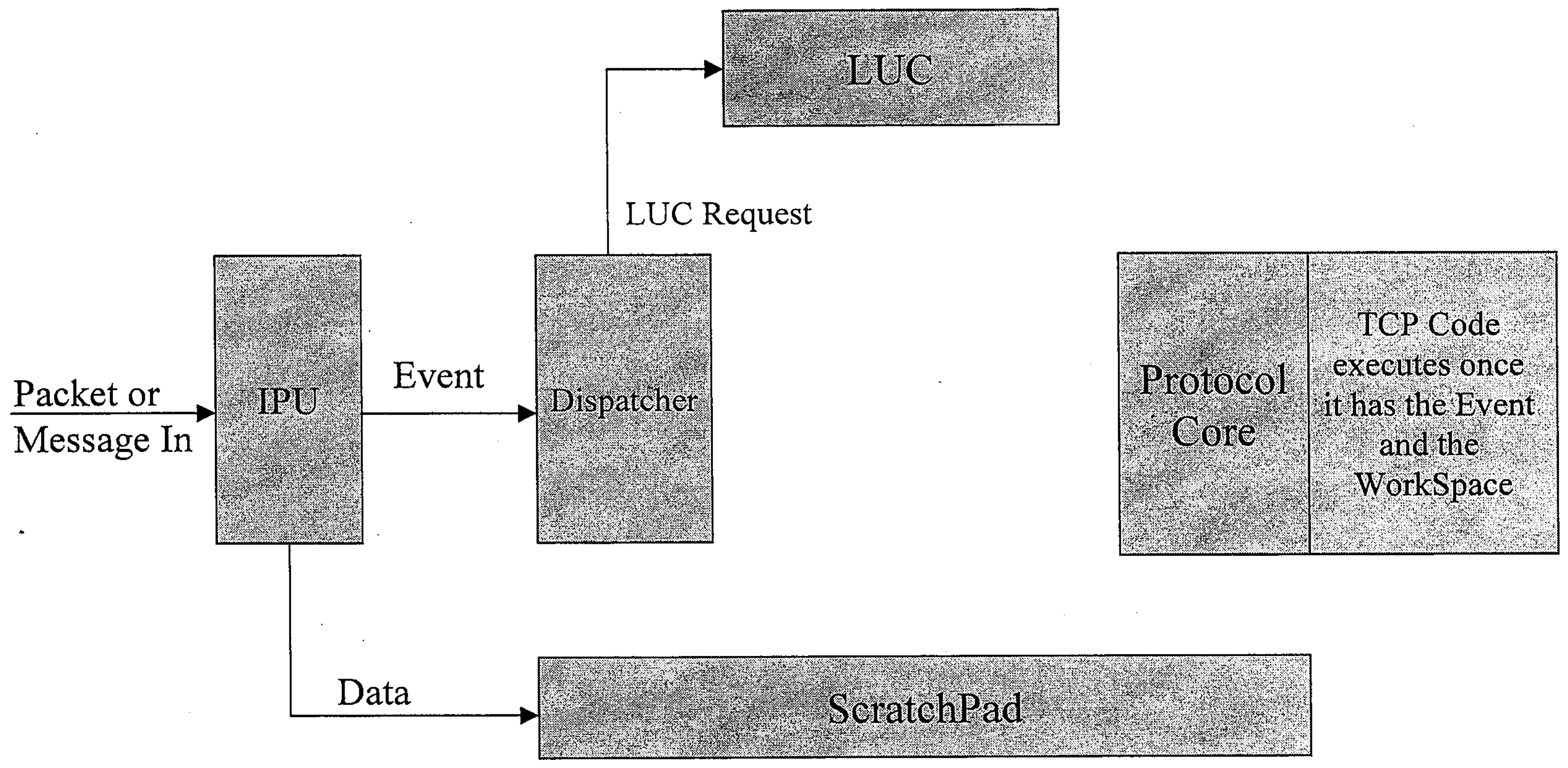
Astute Networks

Pre-TCP DataPath



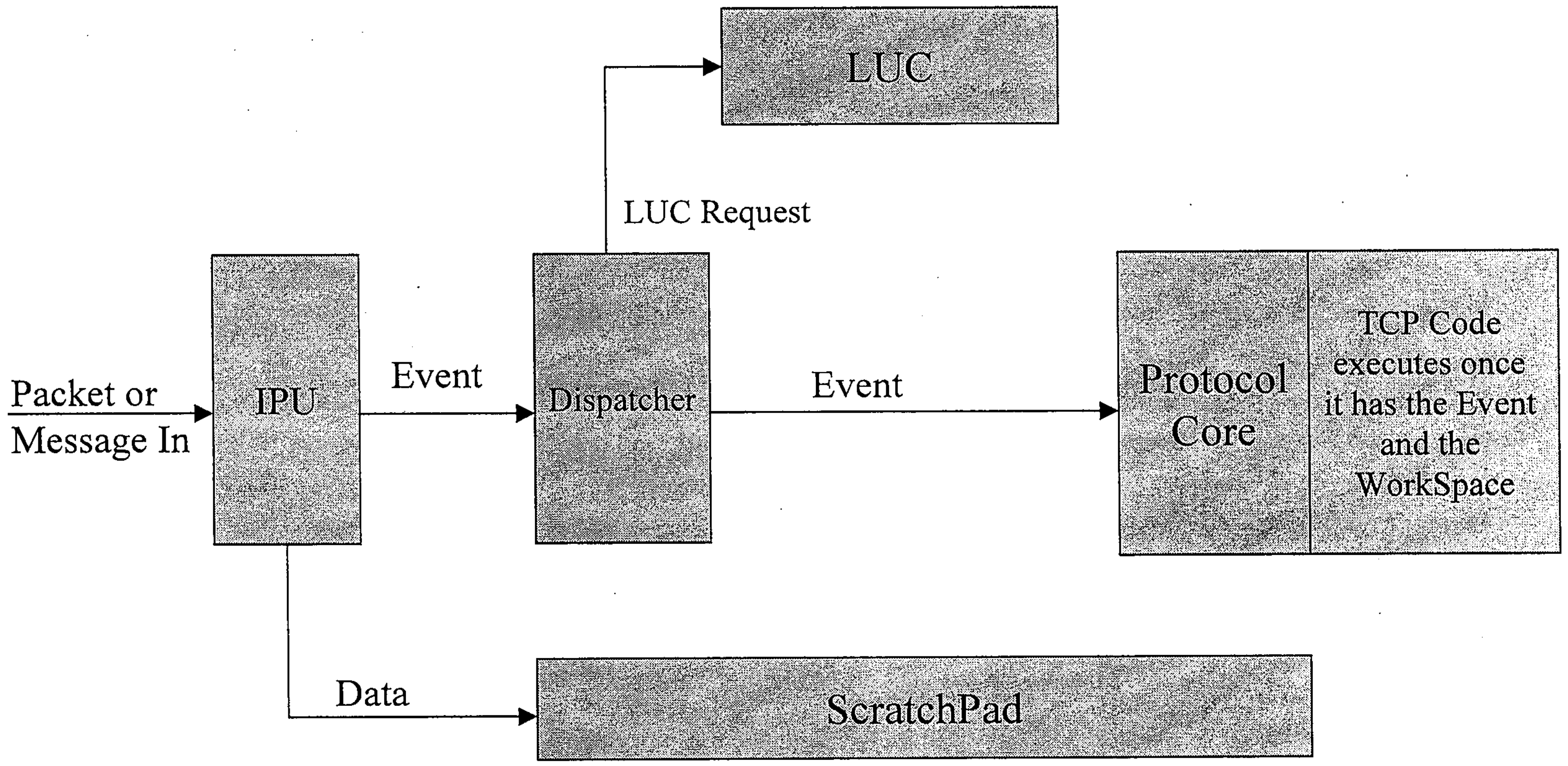
Astute Networks

Pre-TCP DataPath



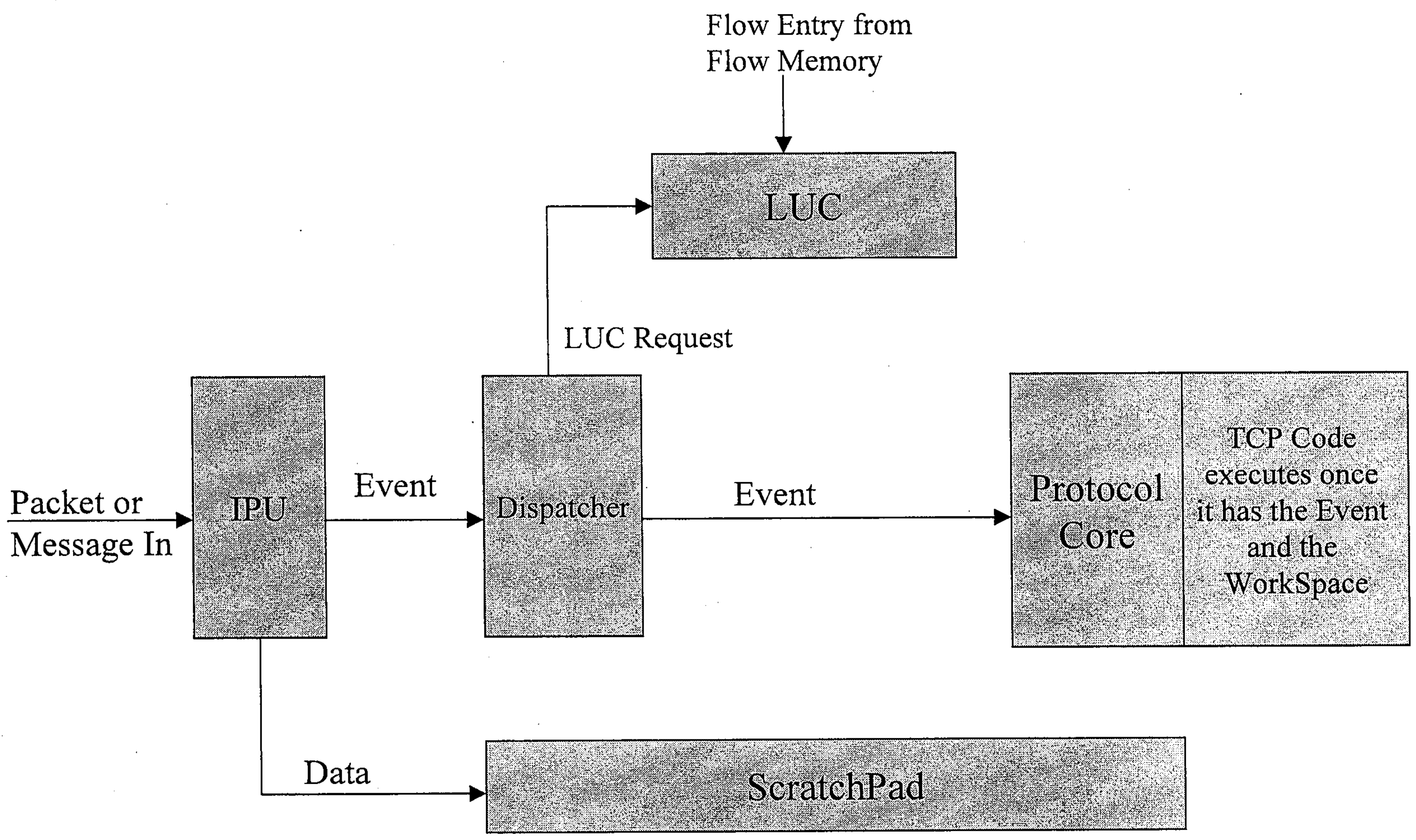
Astute Networks

Pre-TCP DataPath



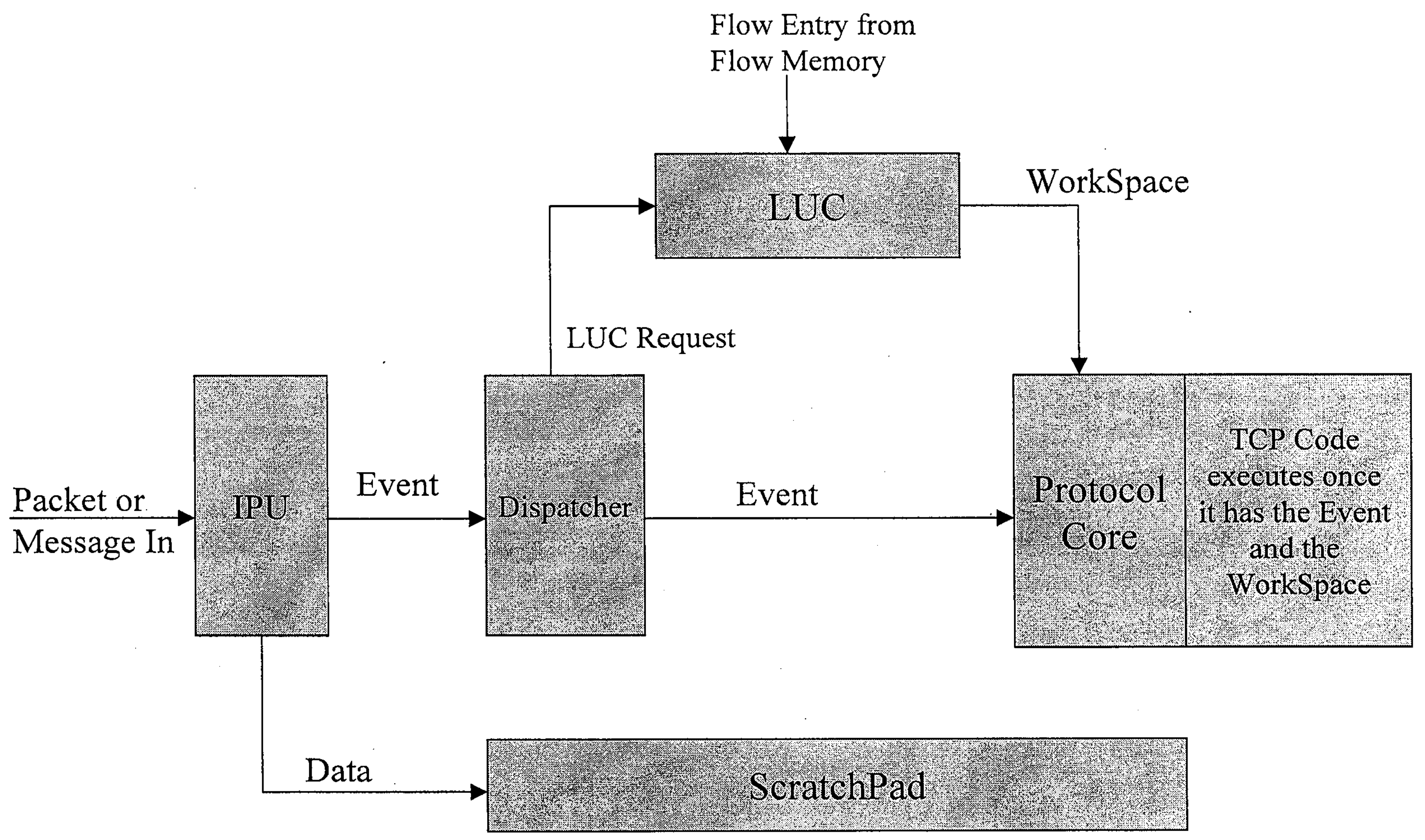
Astute Networks

Pre-TCP DataPath


Flow Entry from Flow Memory
LUC
LUC Request
Packet or Message In
IPU
Event
Dispatcher
Event
Protocol Core
TCP Code executes once it has the Event and the WorkSpace
Data
ScratchPad

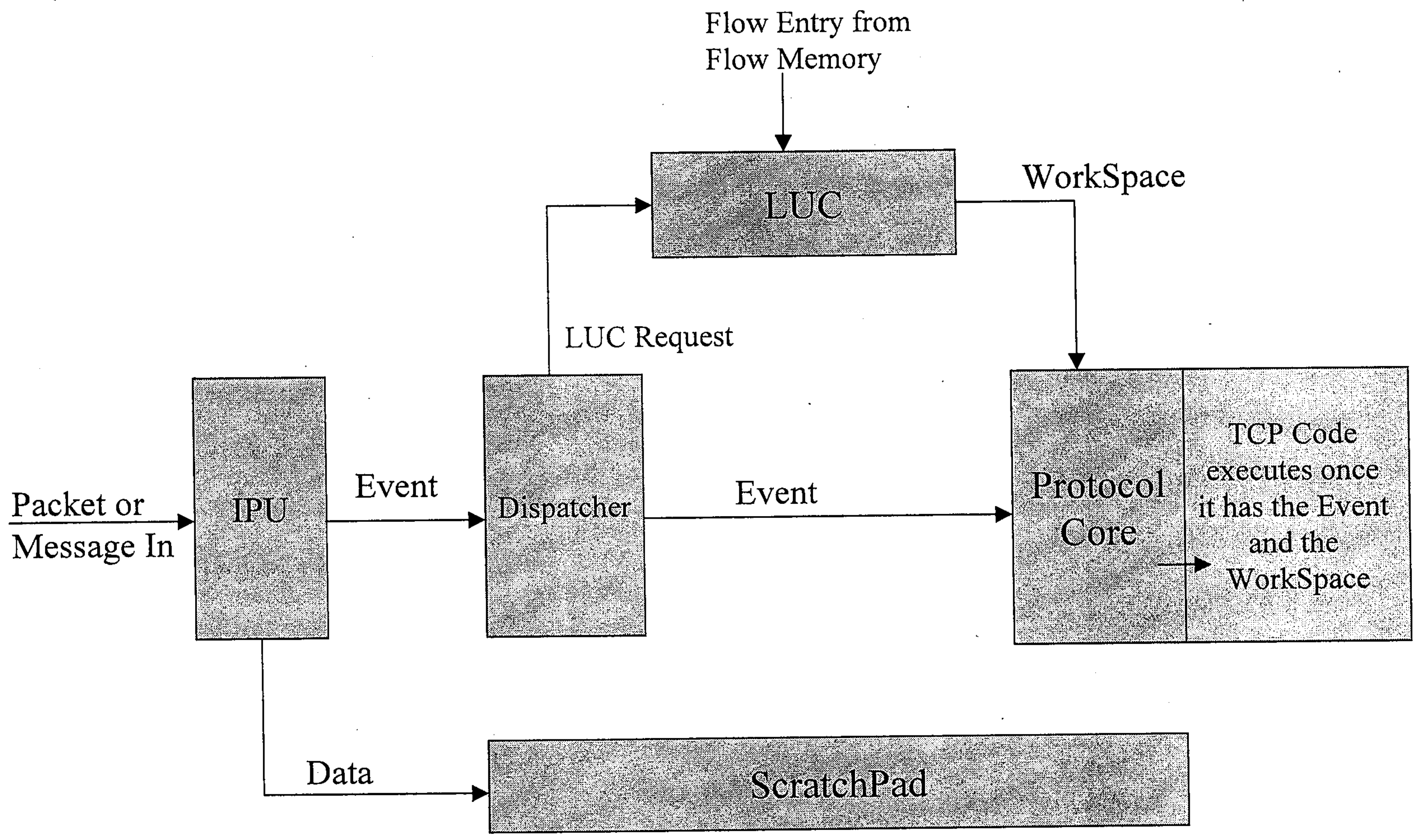
Astute Networks

Pre-TCP DataPath



Astute Networks

Pre-TCP DataPath

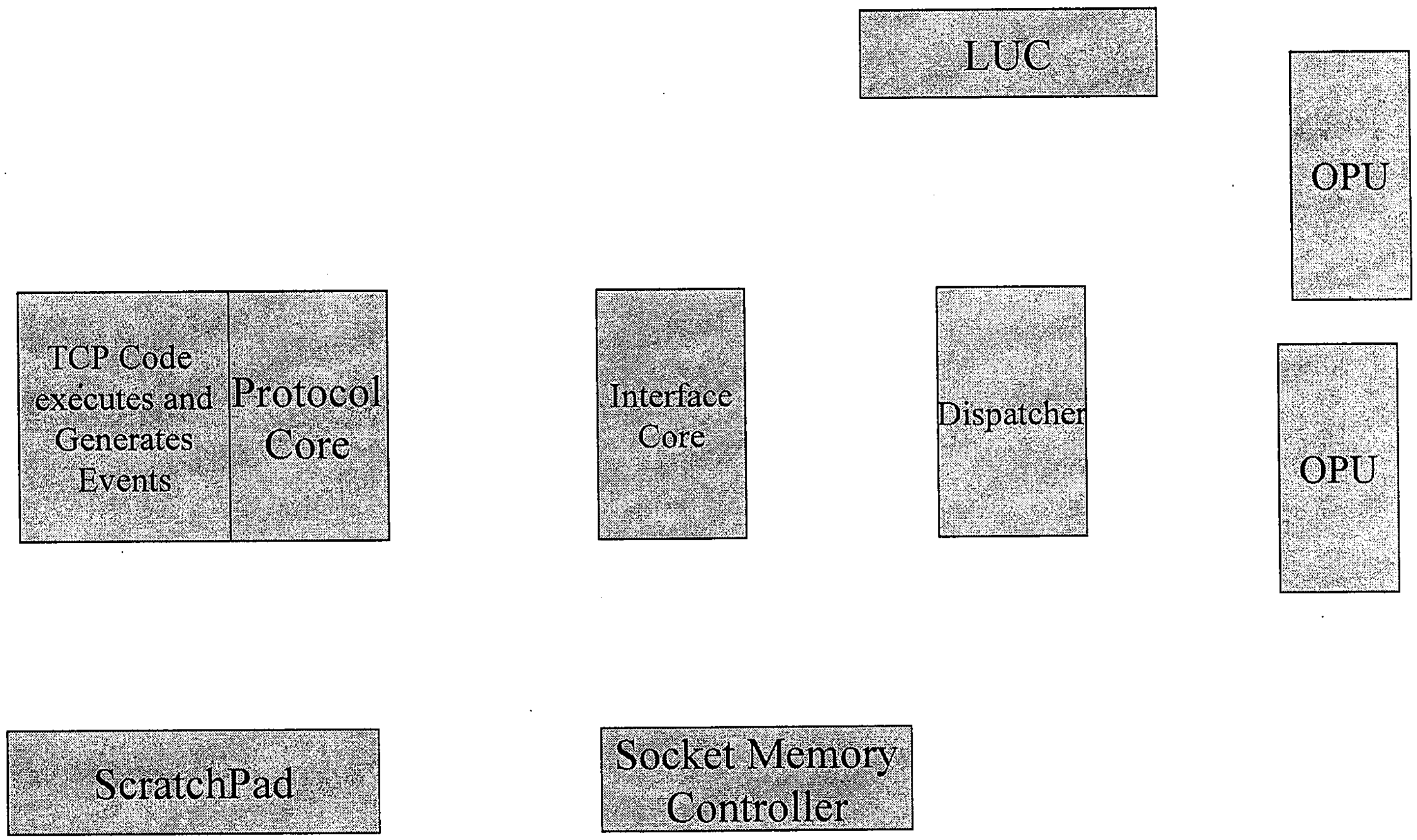


Astute Networks

Astute Networks

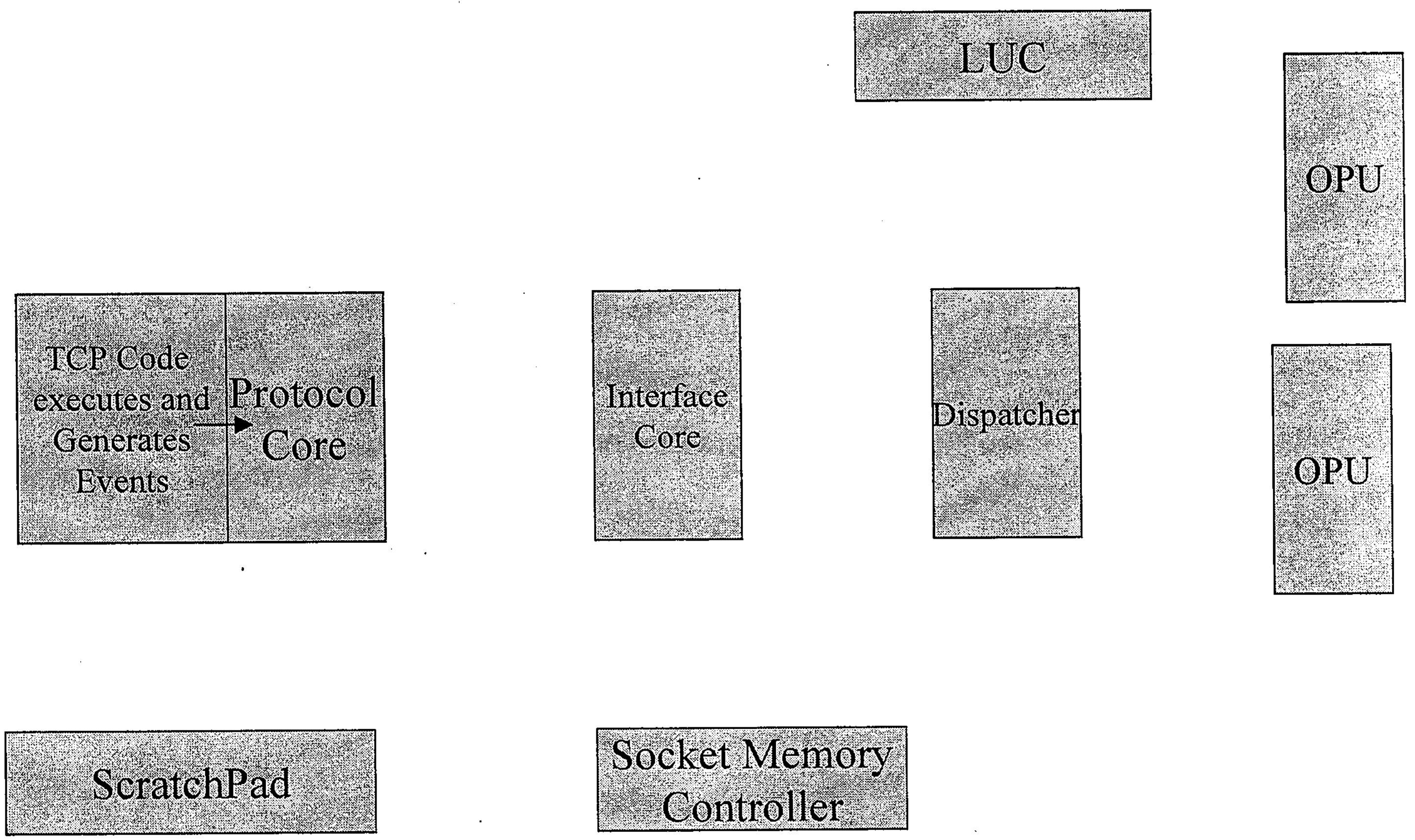
Post-TCP DataPath

Post-TCP DataPath



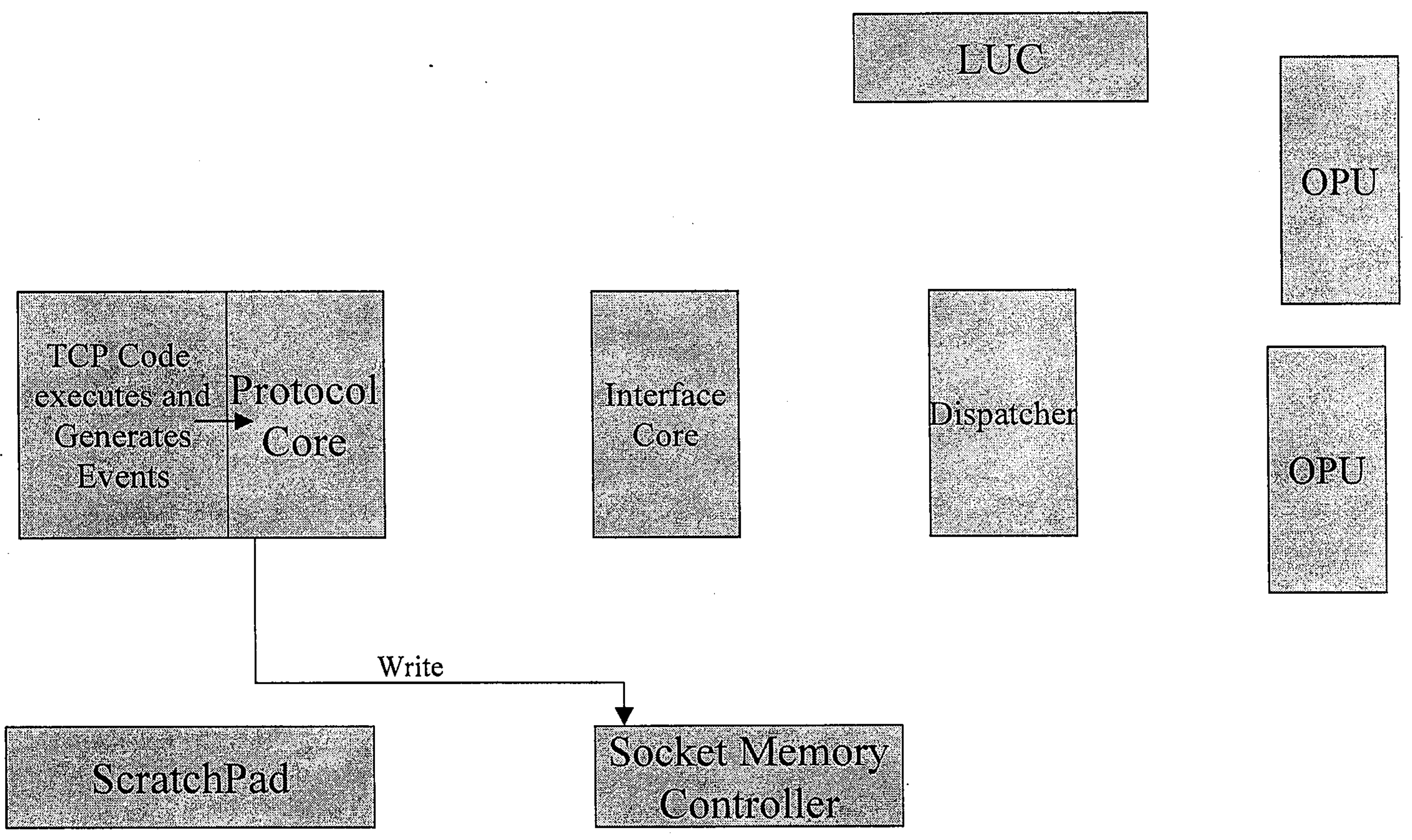
Astute Networks

Post-TCP DataPath



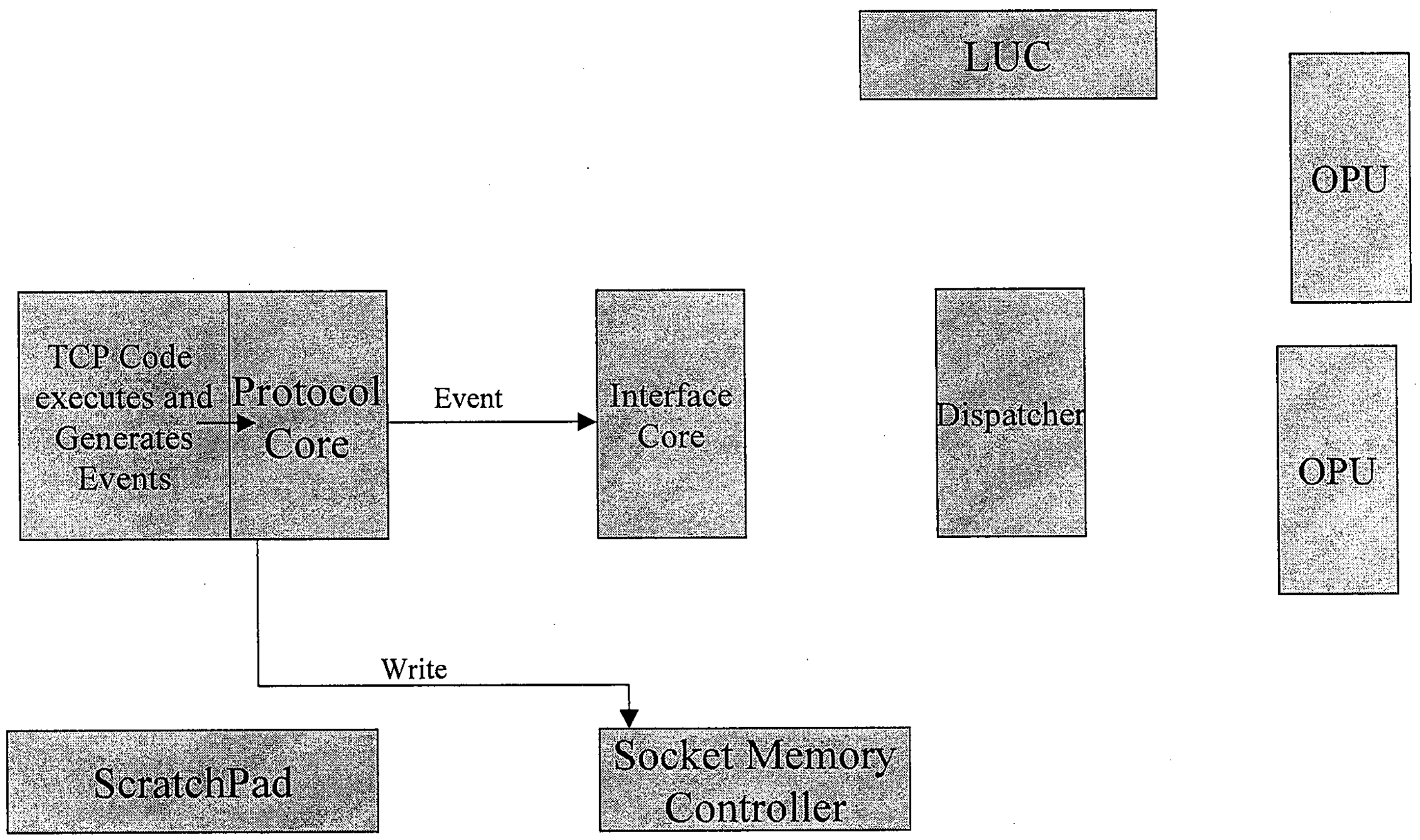
Astute Networks

Post-TCP DataPath



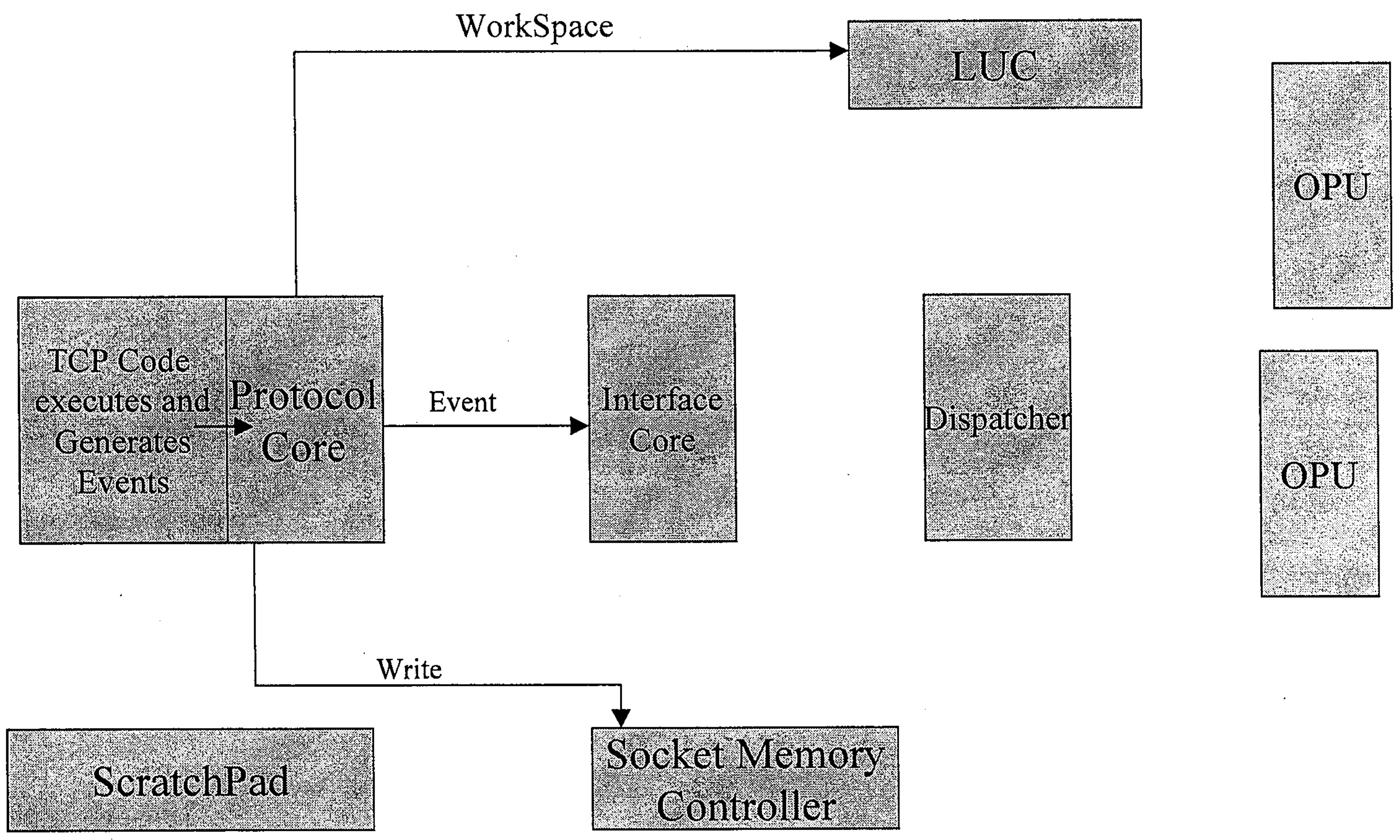
Astute Networks

Post-TCP DataPath



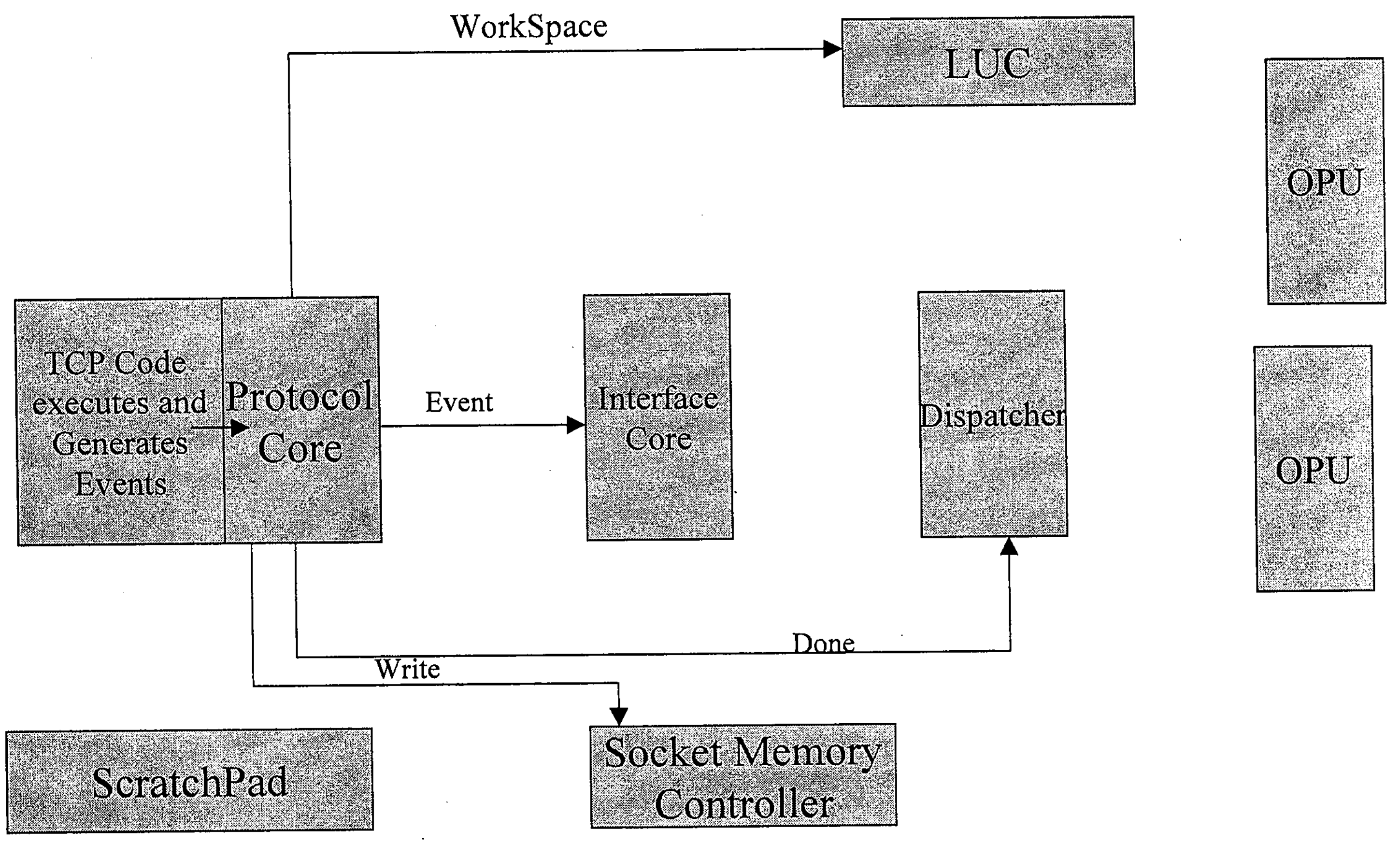
Astute Networks

Post-TCP DataPath



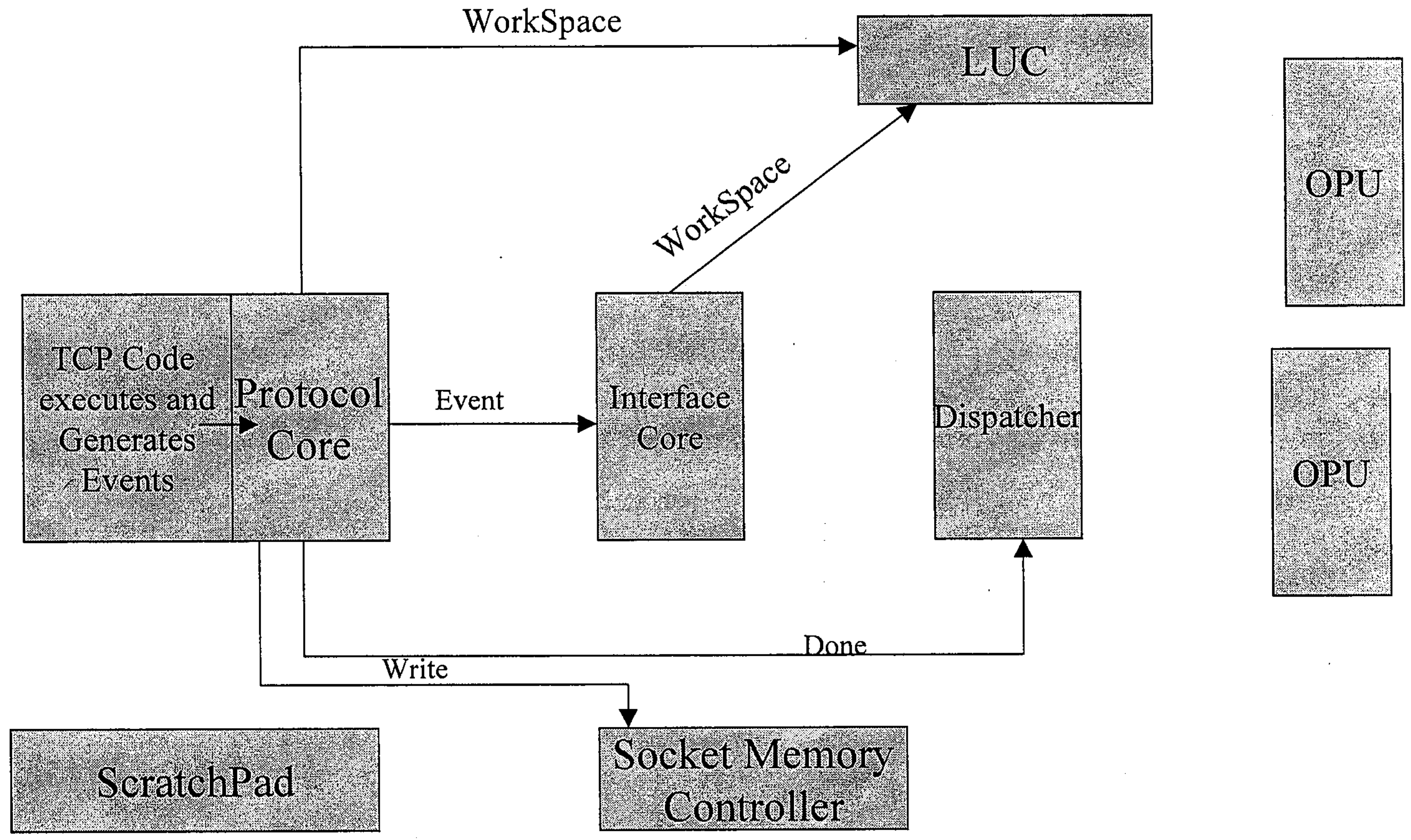
Astute Networks

Post-TCP DataPath



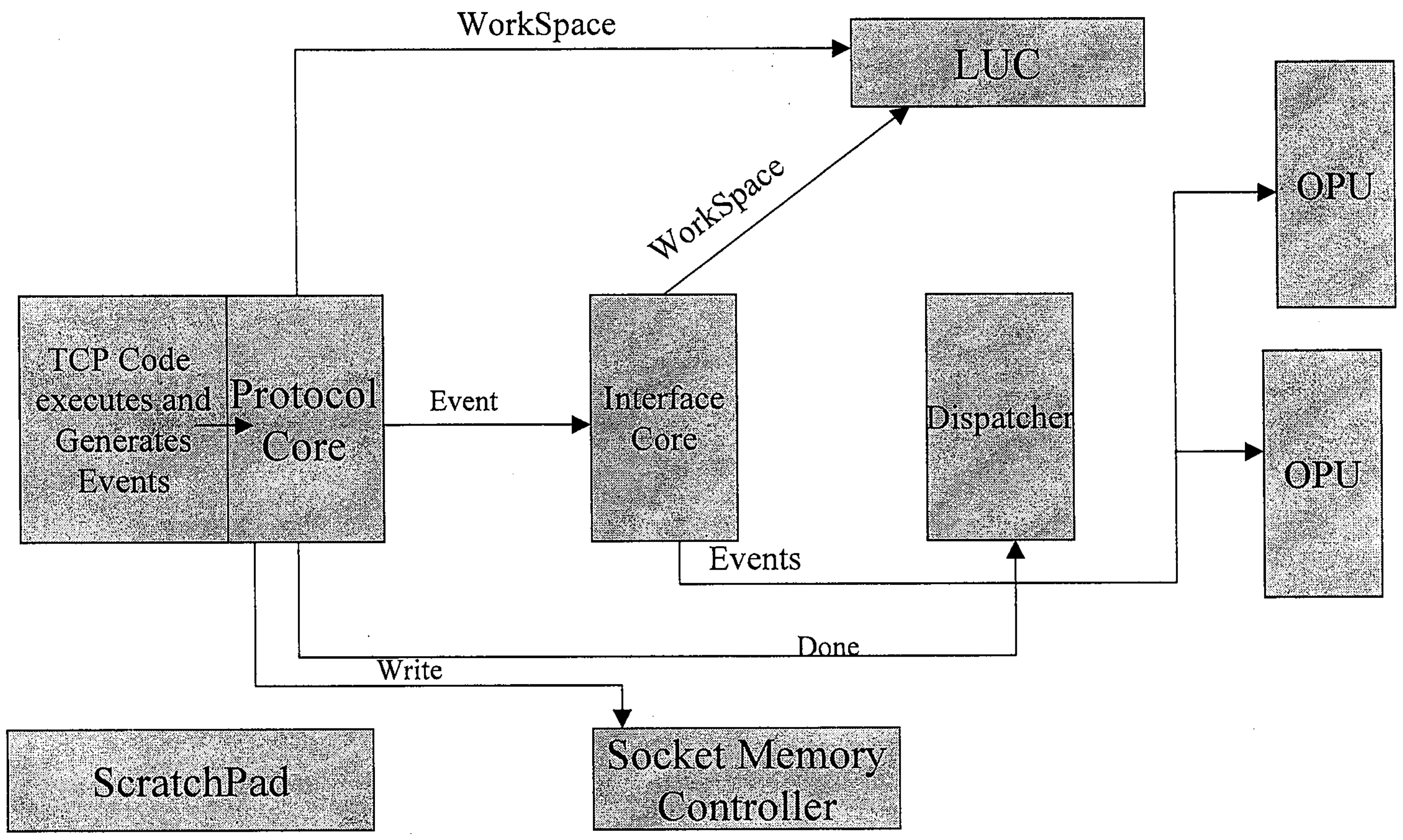
Astute Networks

Post-TCP DataPath



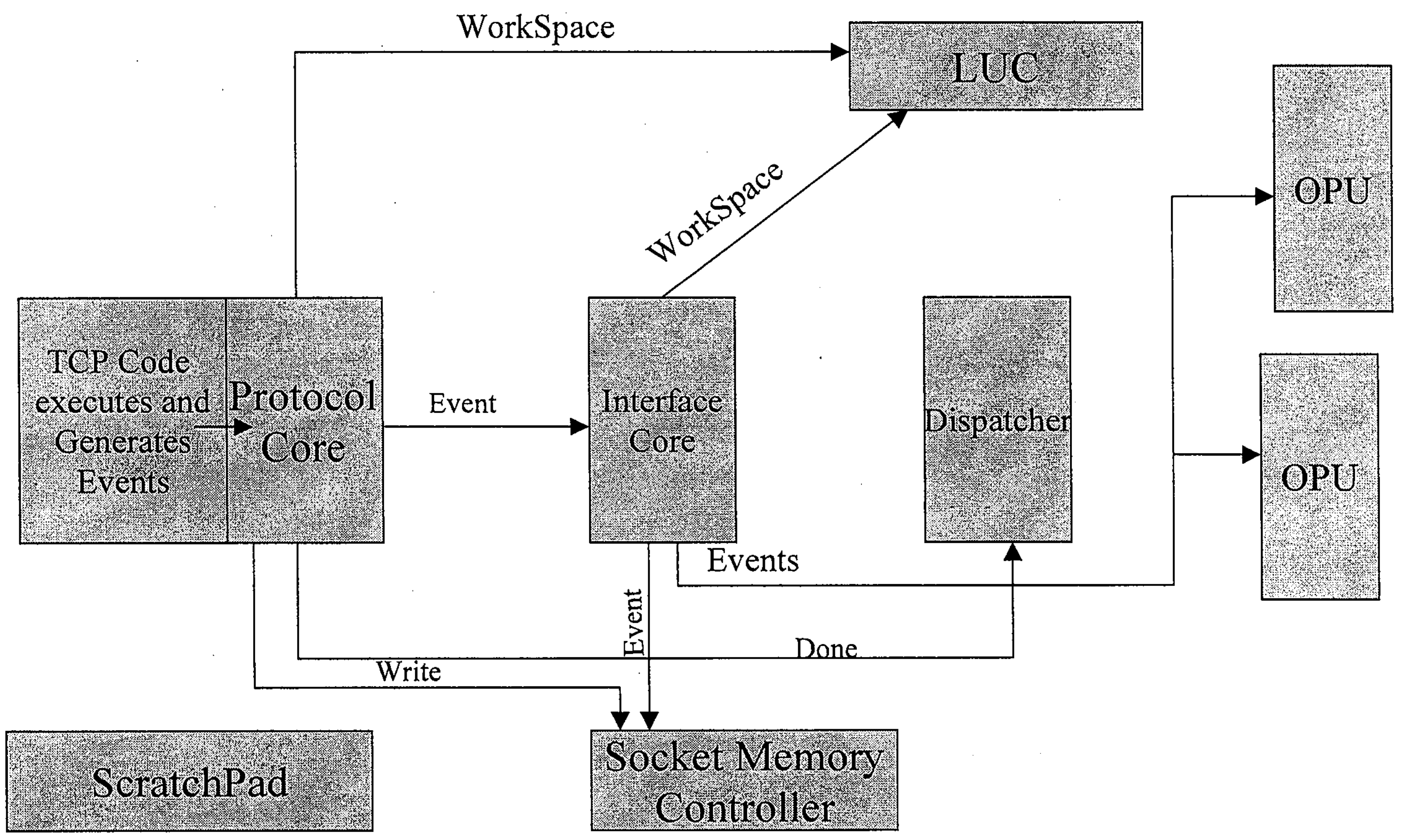
Astute Networks

Post-TCP DataPath



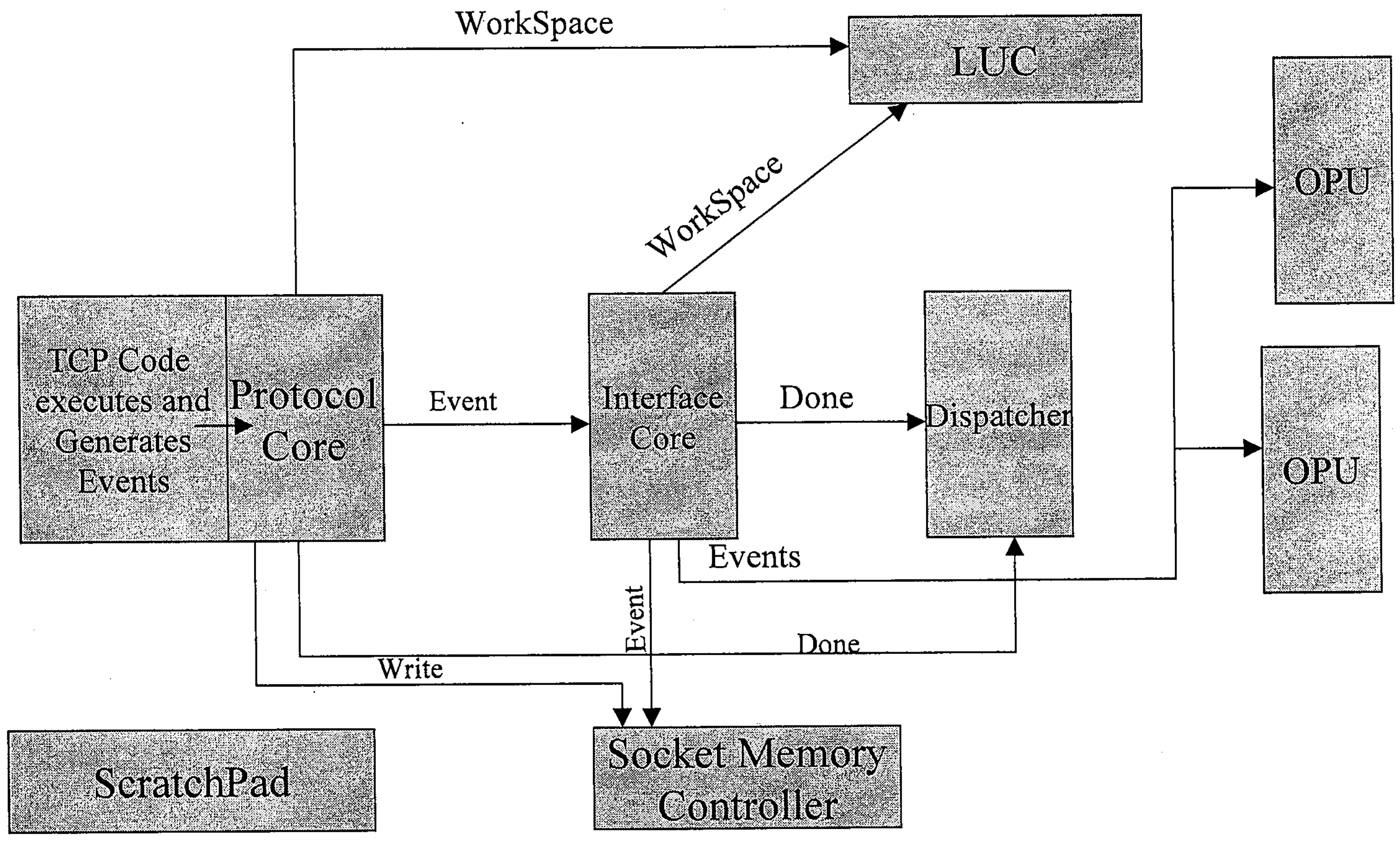
Astute Networks

Post-TCP DataPath



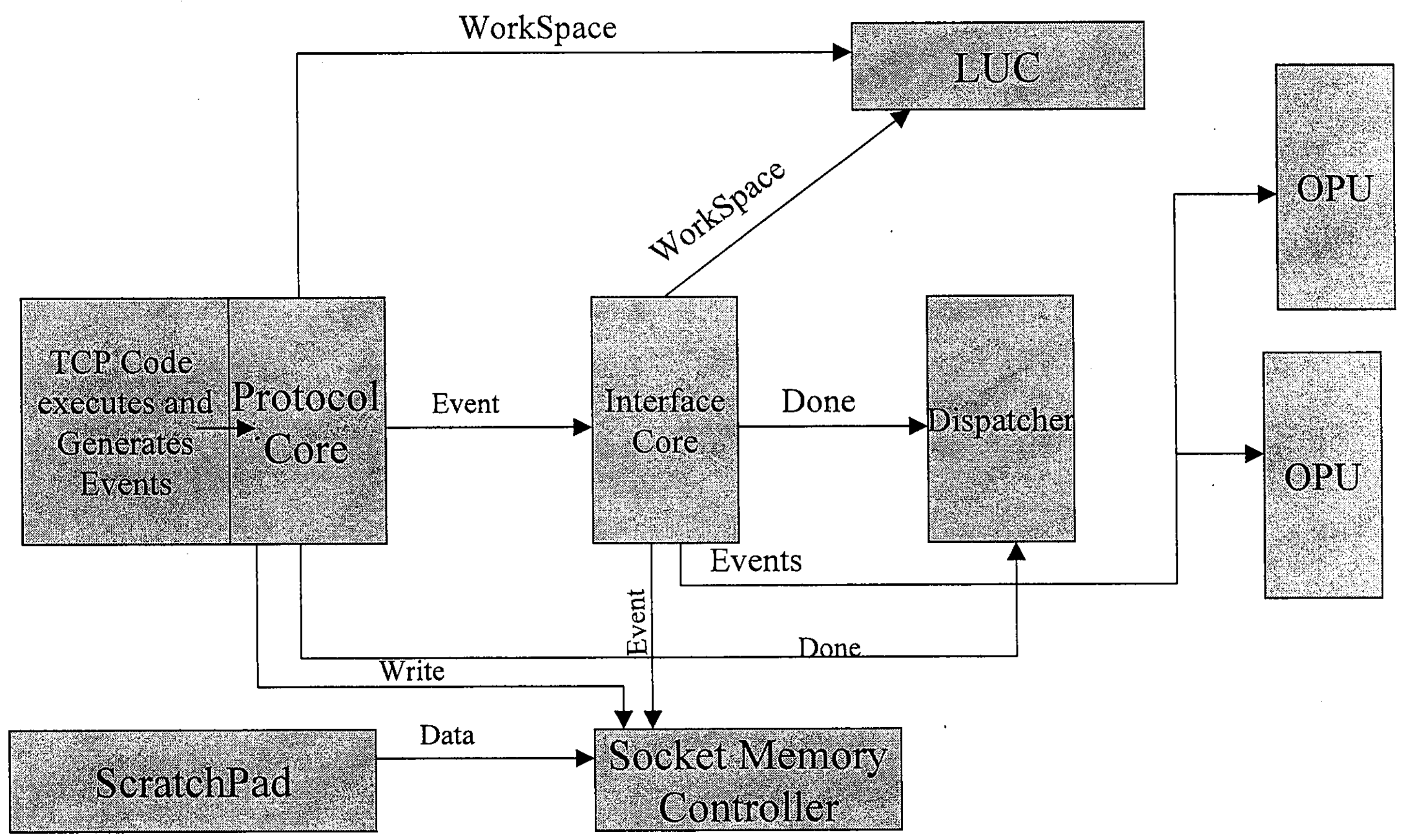
Astute Networks

Post-TCP DataPath



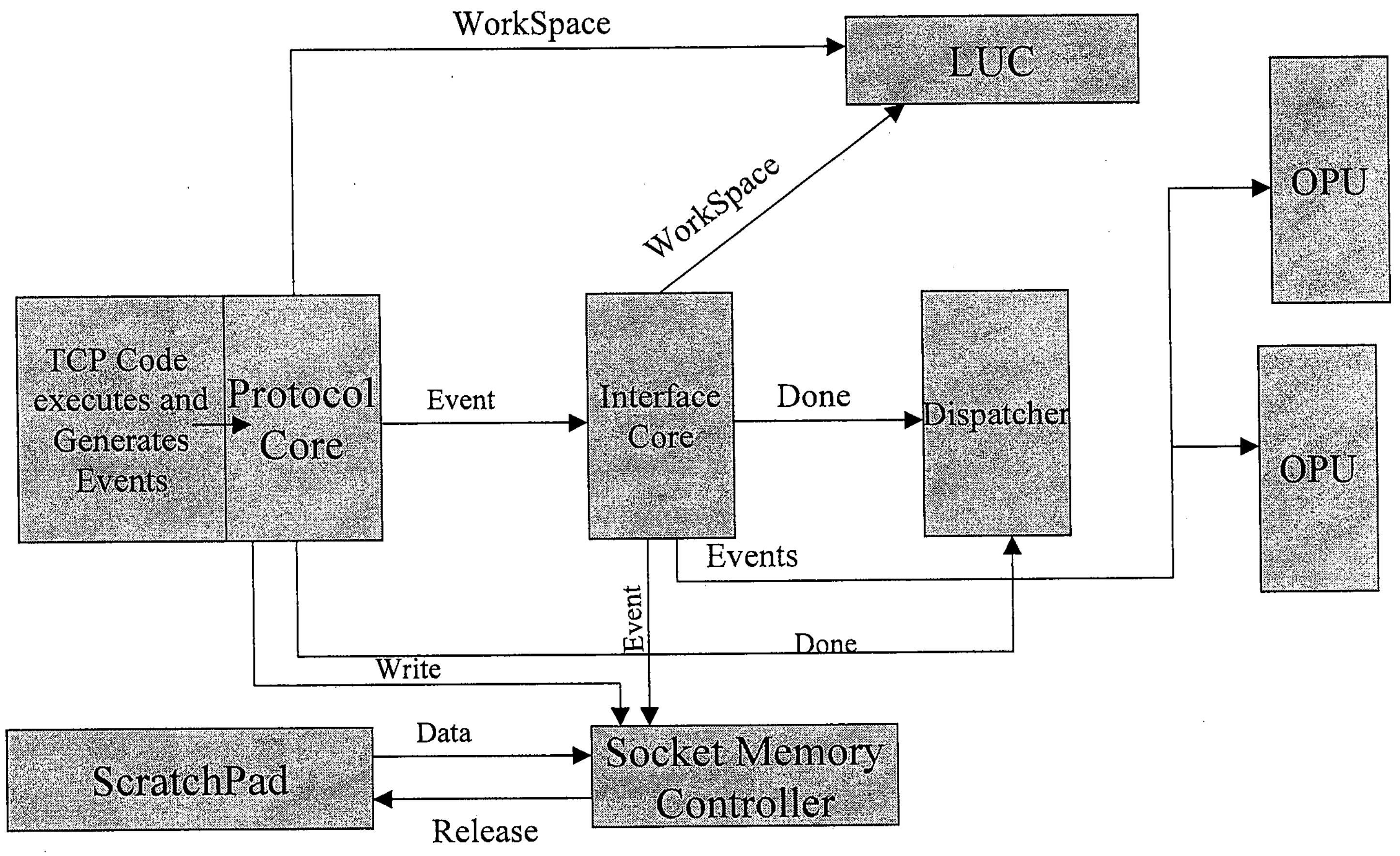
Astute Networks

Post-TCP DataPath



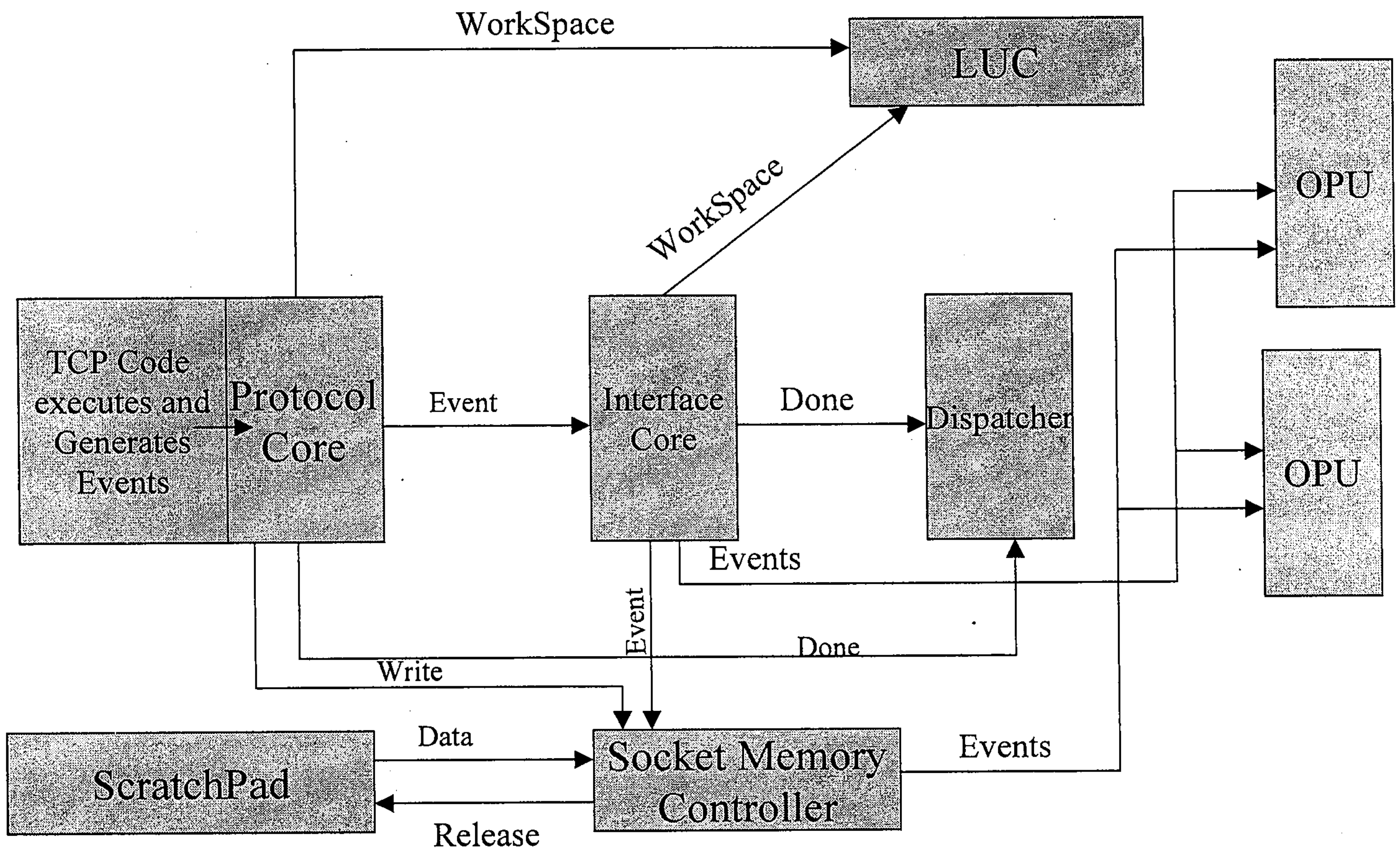
Astute Networks

Post-TCP DataPath



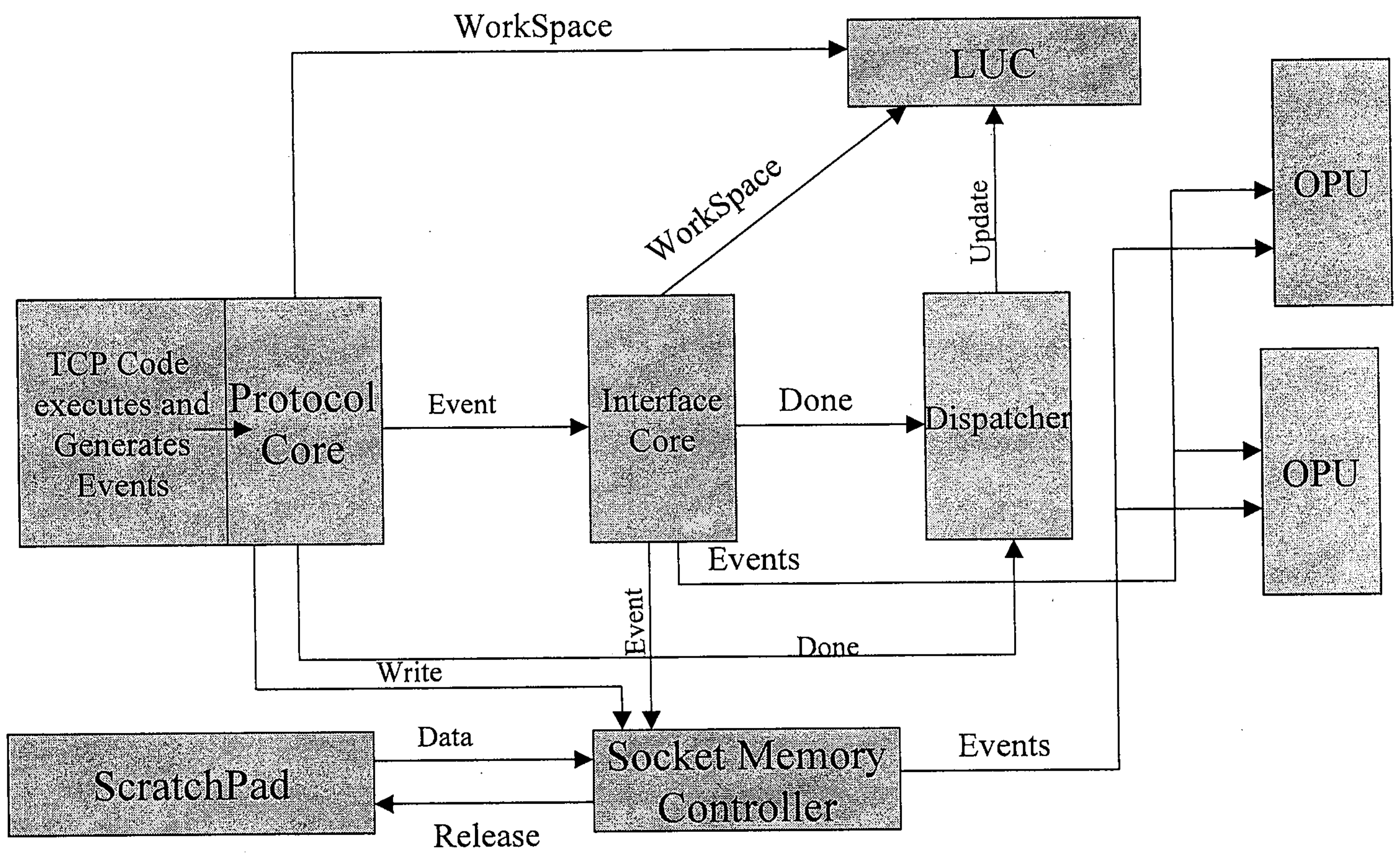
Astute Networks

Post-TCP DataPath



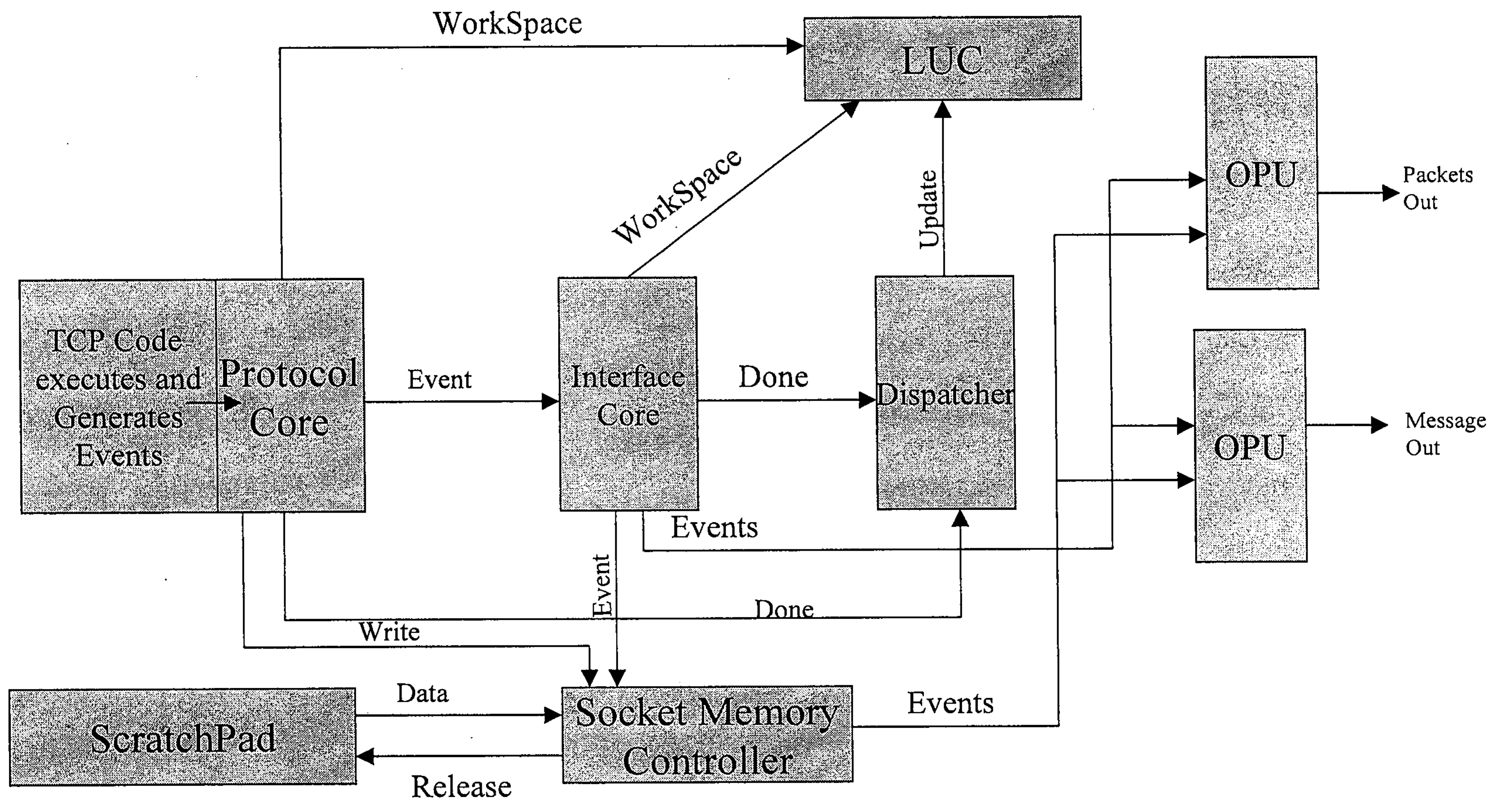
Astute Networks

Post-TCP DataPath



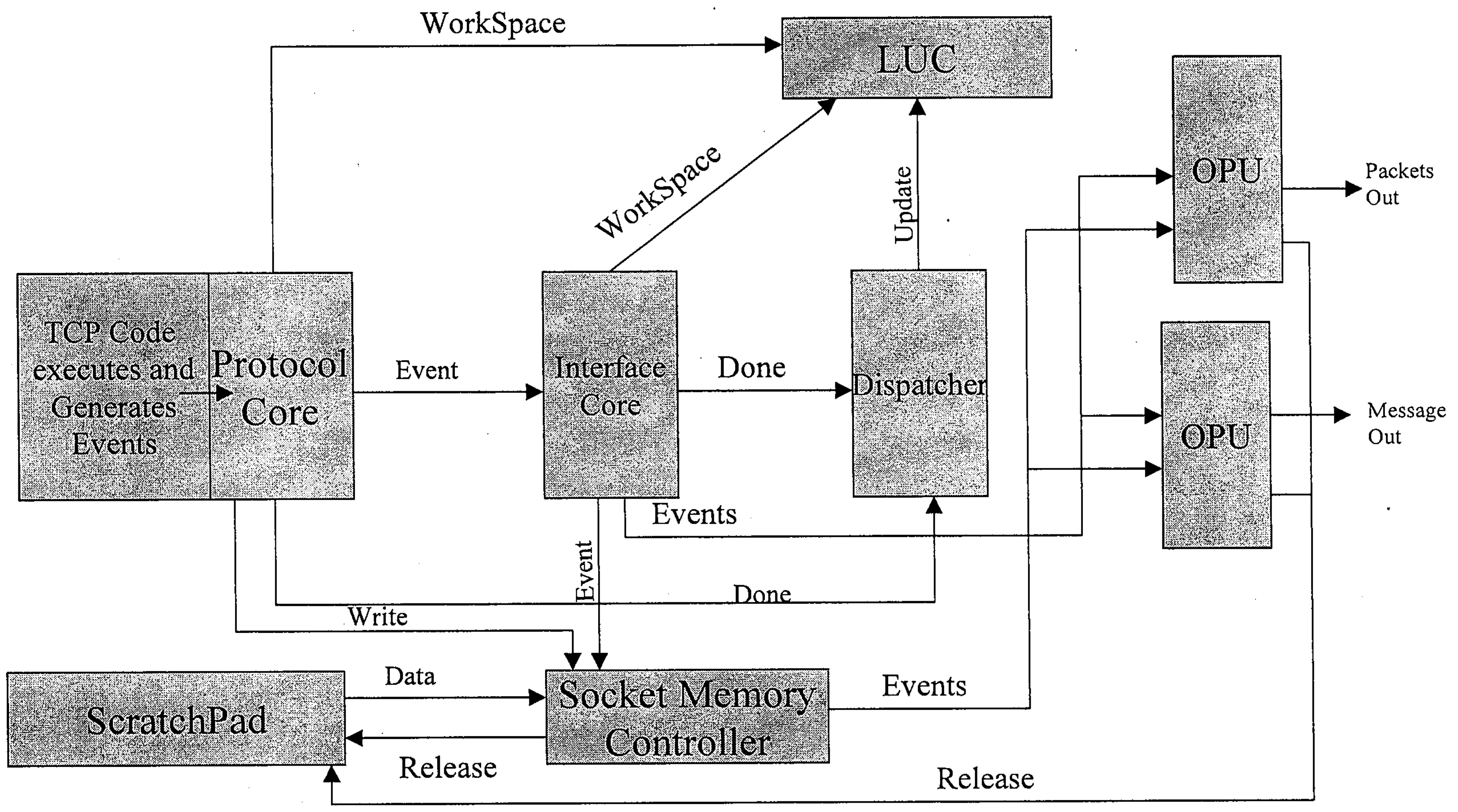
Astute Networks

Post-TCP DataPath



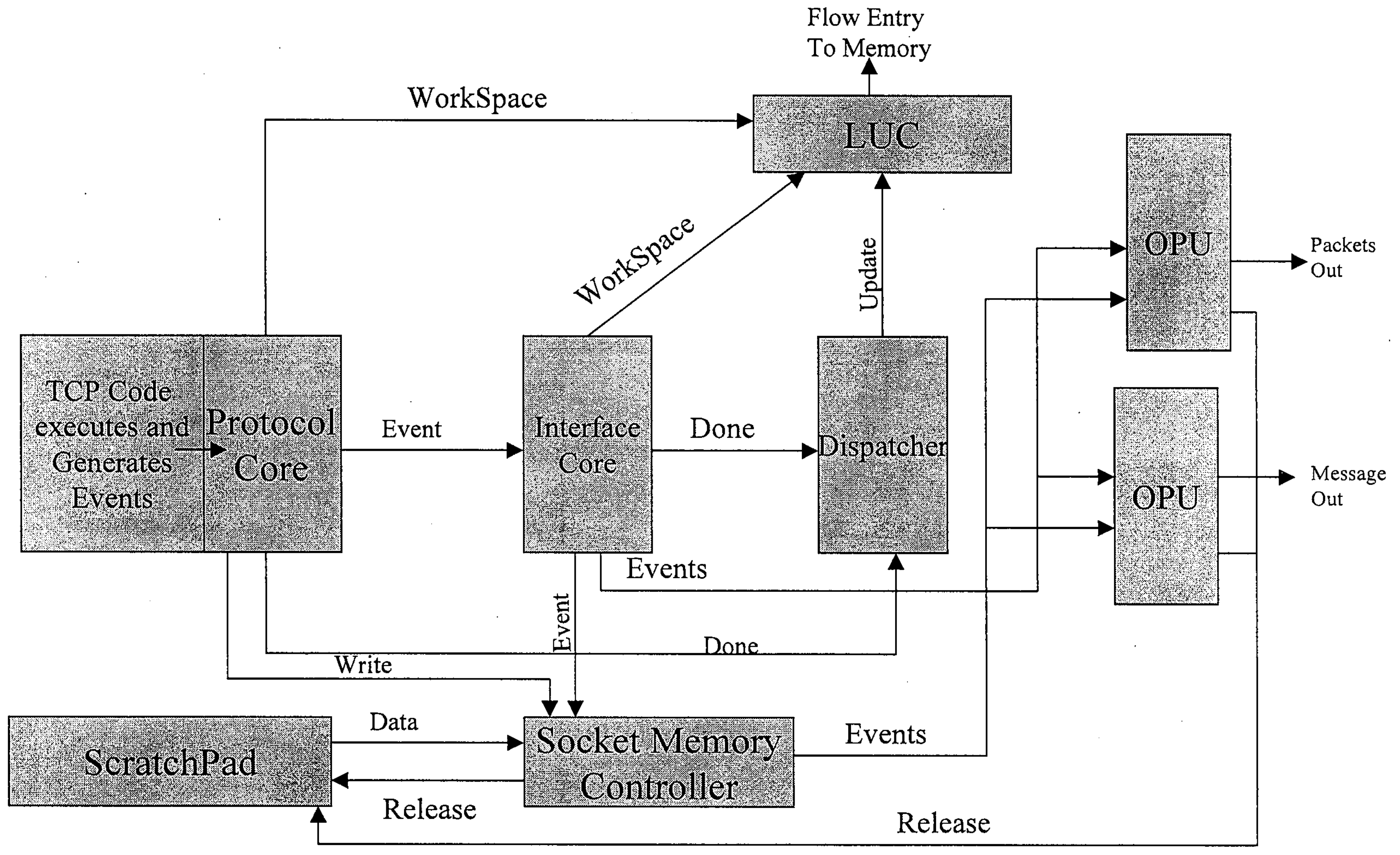
Astute Networks

Post-TCP DataPath



Astute Networks

Post-TCP DataPath



Astute Networks

Why 16 Protocol Cores?

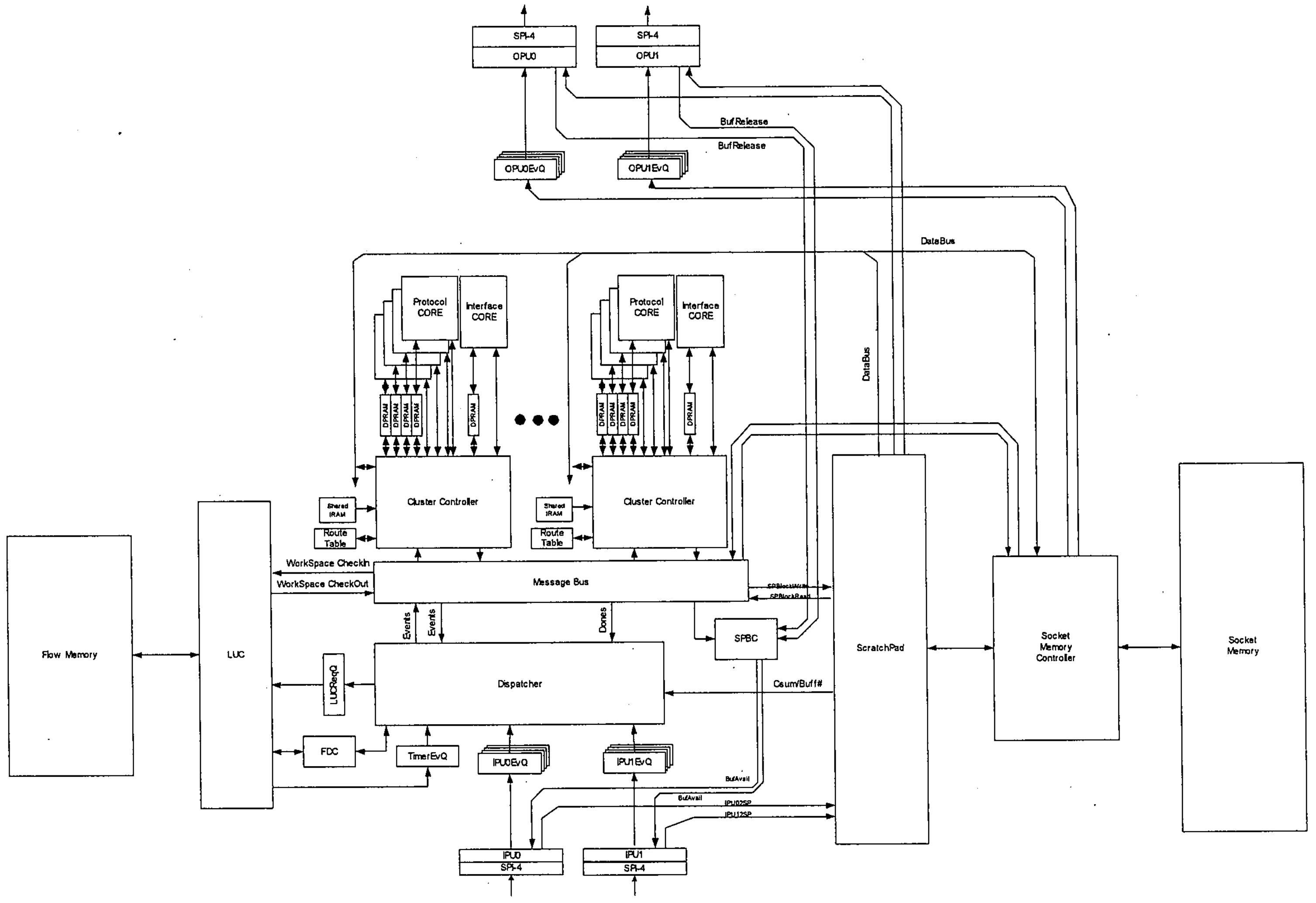
- Analysis of TCP code
  - Our code is based upon FreeBSD
  - Brian and Simon did an analysis of the code based upon our architecture
  - Criteria: normal connection setup and teardown with an HTTP Request/Response
  - Estimate of delays through different paths
  - Document available on Intrastute: TCP Processing
- Queuing model simulation
  - NS2 queuing model will delays for code and LUC
- Result
  - 16 protocol cores will be 75% utilized @ 200 MHz

Astute Networks

Message Bus and Cluster Controller

- Why?
  - Sending Events and Workspaces between 16 Protocol Cores and all the entities they communicate with requires a lot of wide busses.
  - We do not want to complete the architecture and verilog and cannot complete physical design.
- Solution
  - Partition the Processors into Clusters and communicate between Clusters.

TCP ASIC Block Diagram

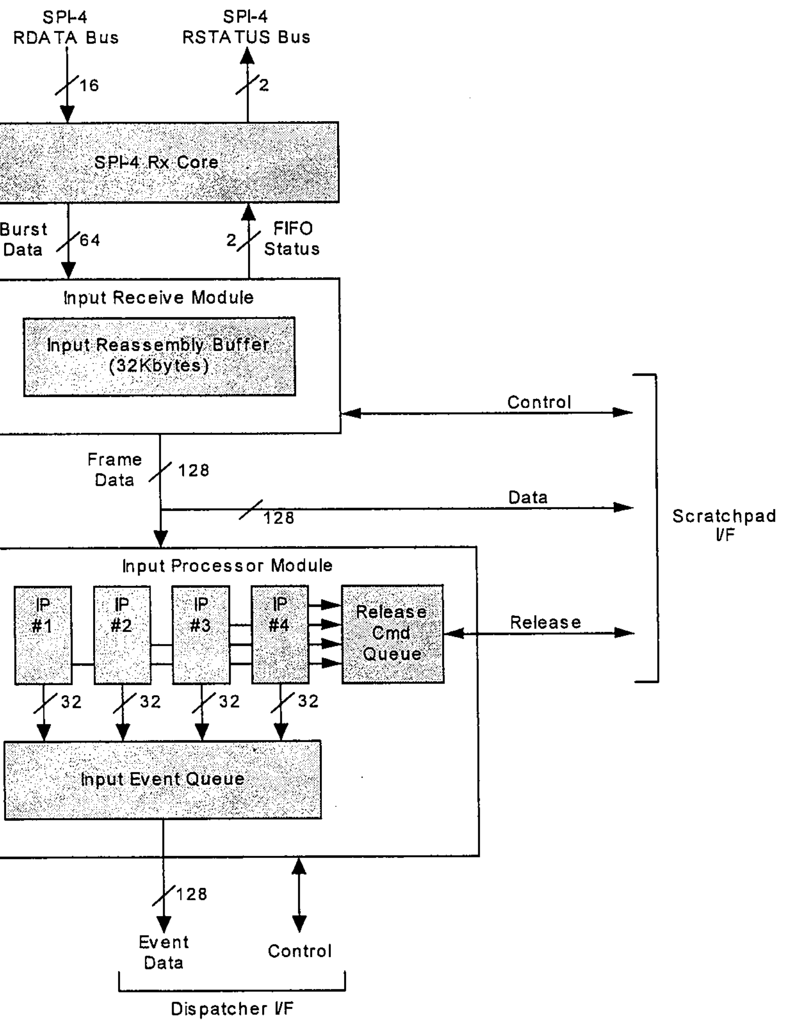


Astute Networks
May, 2001 - 11

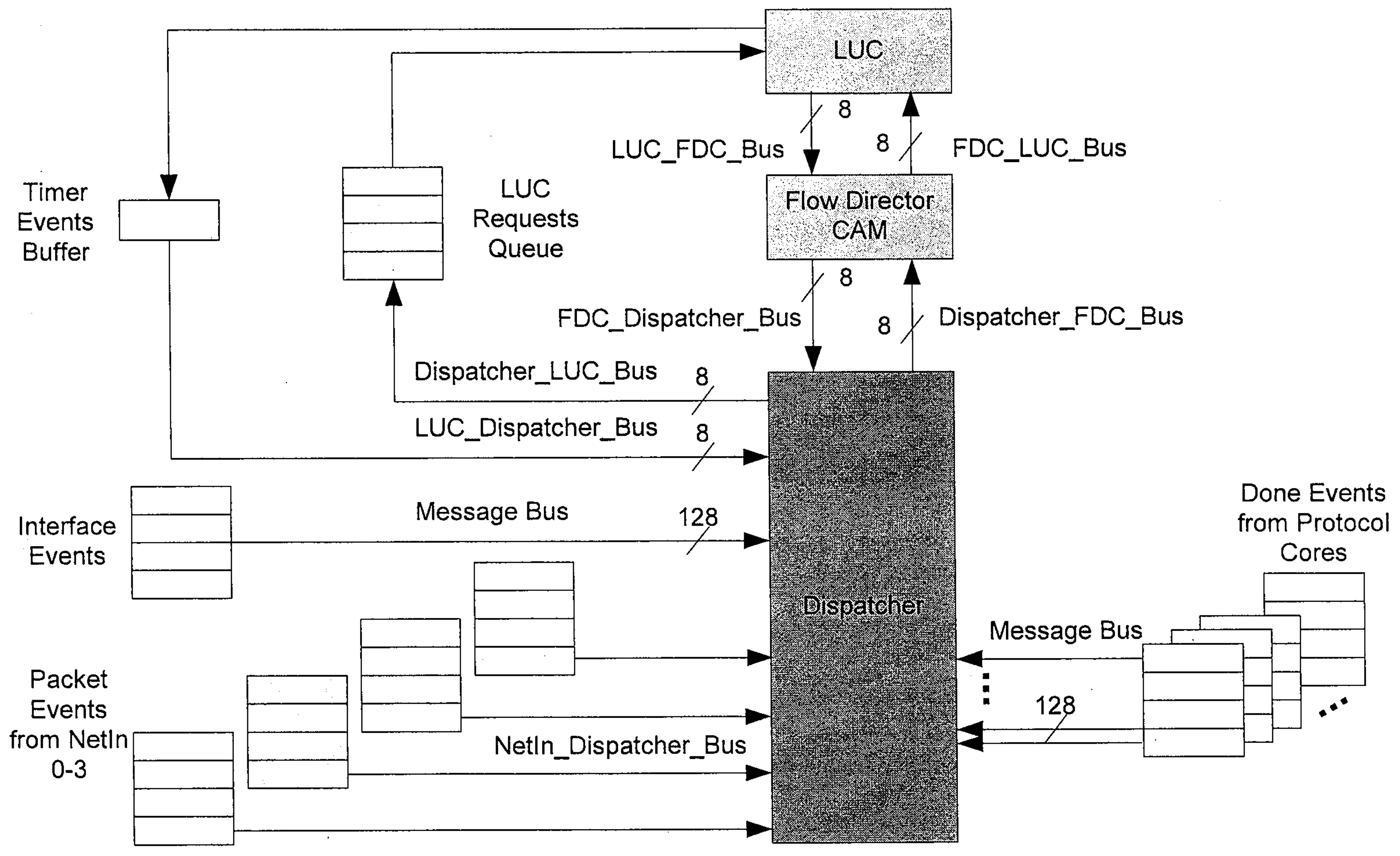
Input Processor Unit

- SPI-4 interface to the external world
  - 10 Gbps Full Duplex
  - Low Pincount
  - High Speed
  - Interfaces to MACs, switching fabrics and more
- Converts Packets or Messages to Events
- Moves Data into ScratchPad
- Contains Packet Processor
  - Programmable
  - Handle different frame formats and headers
  - Up to 256 bytes into packet
- Handles CRC-32 for iSCSI and FCIP

Input Processor Block Diagram



Dispatcher Overview



Astute Networks

Dispatcher Operation

- Takes input events (packet, interface, timer, done):
  - Interacts with FDC to find current protocol core for this flow.
  - Requests a LUC lookup (if required).
  - Passes event onto protocol core.
- Event type determines if Dispatcher requires LUC or does stateless event processing.
- NetIn, interface cores and LUC assign event types.
- Dispatcher registers determine if LUC or stateless processing is needed.

Astute Networks

Event Format

127	V	Command or Response	Event Type	FDC Index	Protocol Core	Work Space	Flow Key [99:96]	
	Flow Key [95:64]							
	Flow Key [63:32]							
	Flow Key [31:0]							0
127	Reserved	Create	Event Size	Event Sub Code	Scratch Data Length			
	Scratch Page Offset			Rsv	Socket ID			
	Scratch Buffer 0			Scratch Buffer 1	Scratch Buffer 2	Scratch Buffer 3		
	Scratch Buffer 4			Scratch Buffer 5	Scratch Buffer 6	Scratch Buffer 7		0
127	Scratch Buffer 8			Scratch Buffer 9	Scratch Buffer 10	Scratch Buffer 11		
	Scratch Buffer 12			Scratch Buffer 13	Scratch Buffer 14	Scratch Buffer 15		
	Scratch Buffer 16			Scratch Buffer 17	Scratch Buffer 18	Scratch Buffer 19		
	Scratch Buffer 20			Scratch Buffer 21	Scratch Buffer 22	Scratch Buffer 23		0
127	Scratch Buffer 24			Scratch Buffer 25	Scratch Buffer 26	Scratch Buffer 27		
	Scratch Buffer 28			Scratch Buffer 29	Scratch Buffer 30	Scratch Buffer 31		
	Event Dependant Data							0

LUC Requests and Timer Events fit in this.

Format fixed in hardware.

This portion of the event is required for anything other than a LUC Request or Timer Event.

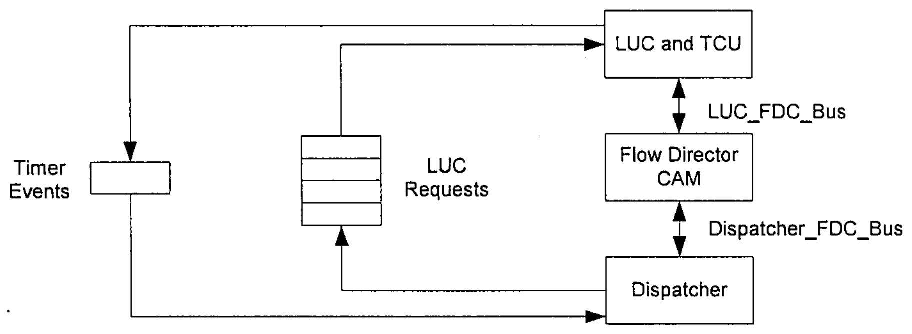
Format fixed in hardware.

Format and size of this part of the event is variable. Software defined.

Astute Networks

FDC Objectives

- Ensures that once a protocol core is assigned to a flow, it processes all outstanding frames for that flow.
- Dispatcher creates entries, LUC deletes them (exception is timers).
- Watch out for timers: must process timer events before packet or interface events.



Astute Networks

FDC Operation

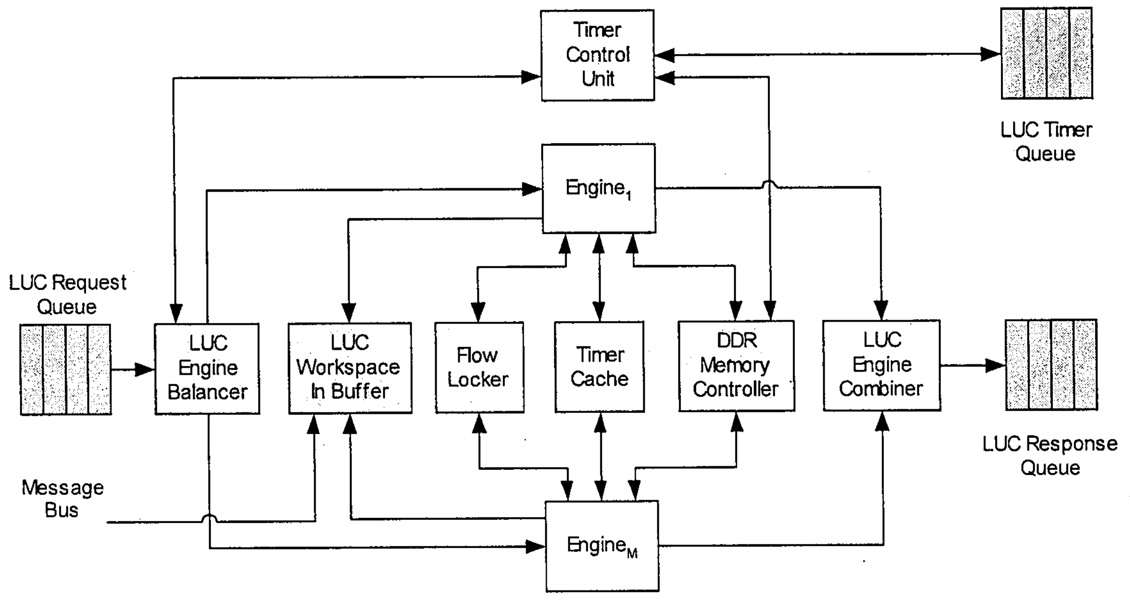
- Uses a CAM to record which protocol core / workspace a flow is assigned to. 64 entries in CAM.
- Keeps track of free/available data structure for spaces in the Event Queue, spaces in the Workspace block.
- Command based via two dedicated buses to LUC and Dispatcher.

Astute Networks

LUC

- Each Flow is uniquely identified by the Flow Key
  - IP Destination and Source Addresses
  - TCP Destination and Source Ports
  - Protocol Type (TCP)
- LUC manages between 16K and 4M Flows
- LUC stores the state of each flow
  - 128 bytes to 2048 bytes
  - TCP state up to 448 bytes
  - Application State at least 64 bytes
- LUC manages Timers
  - 5 TCP Timers
  - 3 Application Timers
- External Random number Input

LUC Block Diagram



Protocol Core

- Xtensa Core
- Instruction Memory – 32 KB
  - Fast path code
- Instruction Cache – 4 KB
  - Slow path code cached from shared cluster memory
- Data Memory – 8 KB
  - Stores workspace, events, stack

Astute Networks

Interface Core

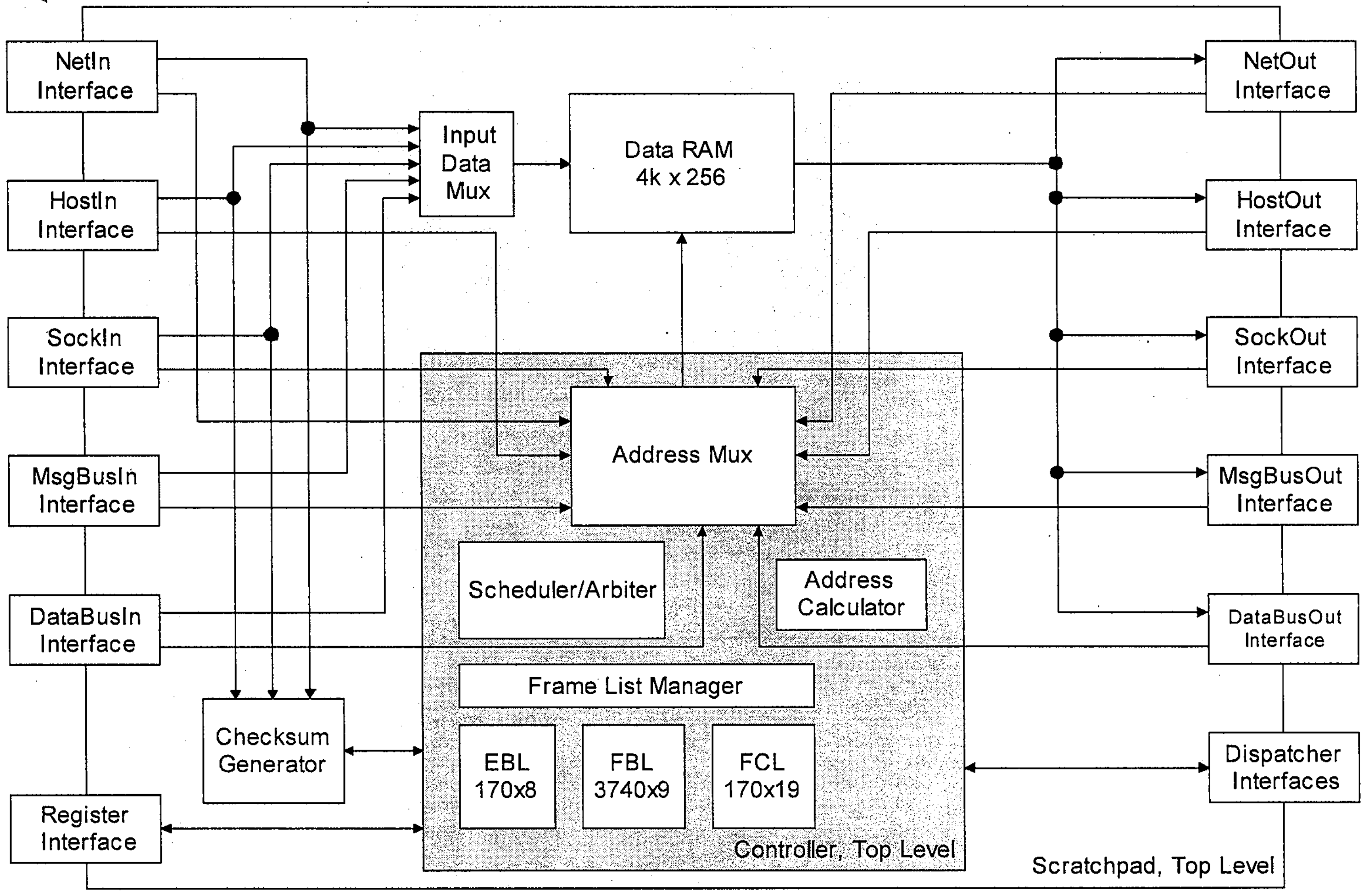
- ◈ Xtensa Core
- ◈ Instruction Memory – 32 KB
  - Fast path code
- ◈ Instruction Cache – 4 KB
  - Slow path code cached from socket memory
- ◈ Data Memory – 8 KB
  - Stores workspace, events, stack

Astute Networks

Cluster Controller

- Connects 4 Protocol Cores and 1 Interface Core to the Message Bus and the DataBus
- Manages cluster instruction memory
- Manages the route lookup table
  - Stores 128 routes

ScratchPad Block Diagram



Astute Networks

Scratchpad Major Components

- Data RAM – 170 pages.
  - Each page is 24x32 bytes or 768 bytes.
- Controller
  - Empty Buffer List, Frame Buffer List, Frame Count List
- Resource Interfaces
  - 5 Input and 5 Output Interfaces

Astute Networks

Socket Memory Controller

- Manages up to 16 GB of Socket Memory
- Used to reassemble frames into an application byte stream and vice versa.
- Provides a simple circular buffer of any size to the Protocol Core
    - Buffer management scheme is hidden
- Gives Application Cores access to storage
    - Instruction memory
    - Data memory

Output Processor Unit

- SPI-4 interface
- Takes Events from Protocol Cores and Interface Cores
- Sends out Packets or Messages out on SPI-4
  - Can send data from the ScratchPad
- Handles CRC-32 for iSCSI and FCIP

Steps in processing an HTTP Request

- **SYN Packet Arrival from Client**
  - Client sends SYN, ACP responds with SYN/ACK
- **Initial ACK Packet from Client**
  - Client responds with ACK, Host is informed of connection
- **HTTP Get from Client**
  - Client sends an HTTP Get
  - Data is written into Socket Memory
  - Host is informed of Data being available
- **Application Read from Host**
  - Host reads the HTTP Get
- **Application Write from Host**
  - Host Writes the HTTP Response
  - Response is sent out to Client
- **FIN from Client**
  - Client closes connection
  - FIN/ACK is sent back to Client
  - Host is informed
- **Application Close from Host**
  - Host closes connection
  - FIN is sent to Client
- **Last ACK from Client**
  - Flow entry is removed

Astute Networks

Syn Packet Arrival from Client

Sender	Receiver	Description	Message Bus	#bytes
		Syn Packet Arrival from Client		
NetIn	Dispatcher	Syn Packet arrives into NetIn. NetIn converts this into an Event that is sent to the Dispatcher.	N	
Dispatcher	LUC	After checking in the FDC, the Dispatcher sends a LookUp Request to the LUC.	N	
Dispatcher	PCn	Based upon FDC results, Event is assigned to Protocol Core PCn and its associated Interface Core Icm. Dispatcher sends Event to PCn.	Y	256
LUC	PCn	After processing LookUp Request from Dispatcher, LUC sends Protocol WorkSpace to PCn	Y	512
LUC	ICm	LUC then sends Application WorkSpace to ICm	Y	
PCn	ICm	PCn gets the Syn Event and associated Workspace. PCn sends Event to Icm.	N	
PCn	NetOut	After processing the Syn, PCn send Event to NetOut with a SYN/ACK.	Y	256
PCn	SMC	Since this is a new flow, PCn sends a Create Socket Memory Buffers to SMC	Y	
PCn	Dispatcher	Finally, PCn send a Done message to Dispatcher	Y	16
PCn	LUC	PCn sends Workspace back to LUC	Y	512
ICm	LUC	ICm also sends a WorkSpace to LUC	Y	
ICm	Dispatcher	ICm sends a Done to Dispatcher	Y	16
Dispatcher	LUC	Dispatcher sends an Update to LUC	N	
LUC	PCn	When it is done writing the WorkSpace to the Flow Table, LUC sends to PCn the command to clear Valid bit		
LUC	ICm	When it is done writing the WorkSpace to the Flow Table, LUC sends to ICm the command to clear Valid bit		

Astute Networks

Application Support

- HTTP Proxying and Splicing
- External SSL Accelerator
- iSCSI and FCIP

Astute Networks

HTTP Proxying and Splicing

- ACP terminates connection to client
- For HTTP 1.1. passes the request to Host
- Host responds with an Open connection to Server1
- ACP opens connection to Server1
- ACP splices Client connection to Server1 connection
- Next Request gets sent to Host
- Host responds with an Open connection to Server2
- ACP opens connection to Server2
- ACP splices Client Connection to Server2 connection AFTER Server1 data is done

iSCSI, FCIP, SSL

- All these protocols are handled in a similar fashion by the ACP.
- When a packet is received, after being processed by the Protocol Core, the Interface Core can peek at its contents and maintain its own state of that flow
  - e.g. look at the SSL record header to determine length

Astute Networks

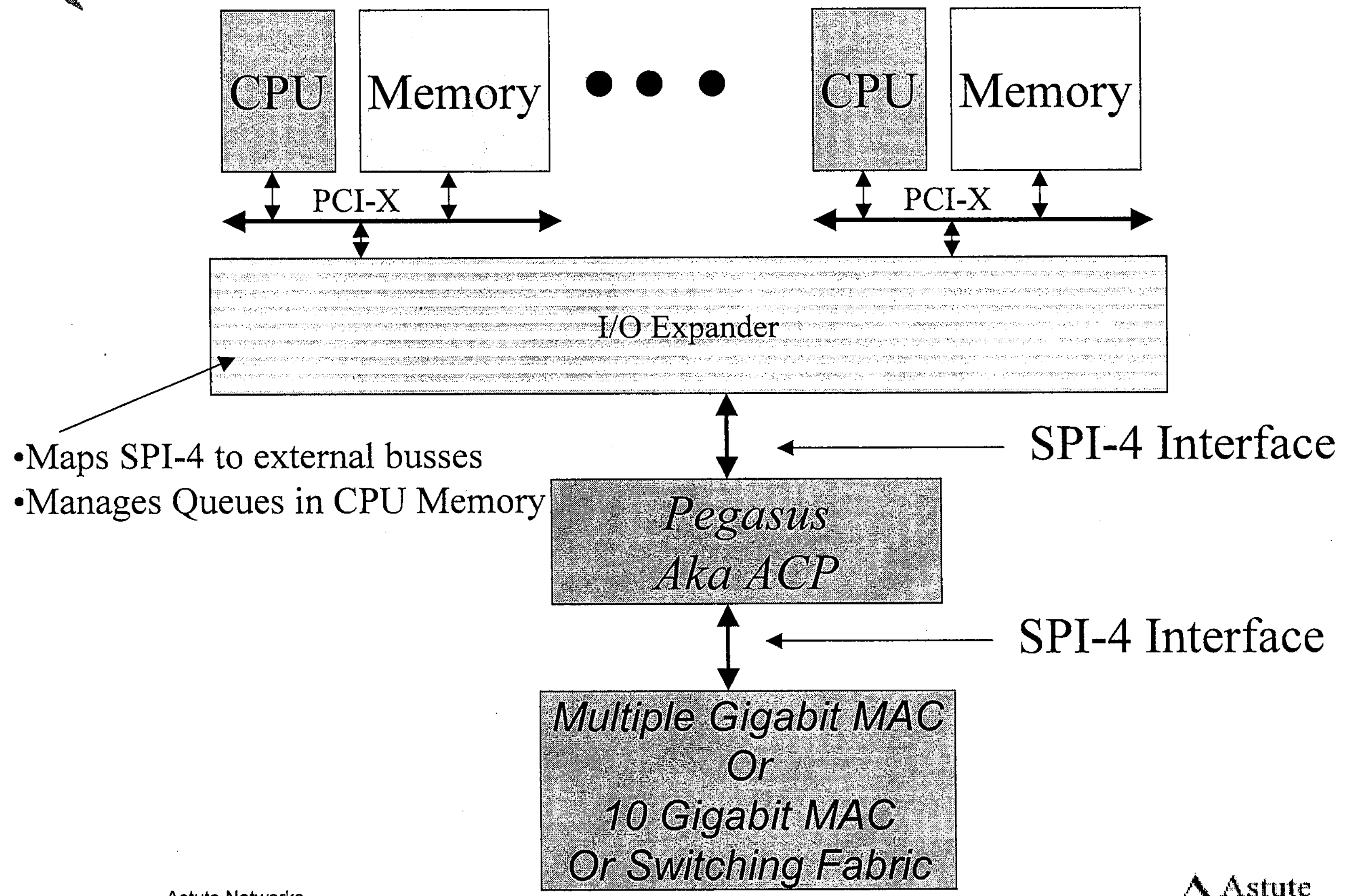
iSCSI and FCIP

- These two protocols have similar requirements
  - Verify the PDU via CRC-32
  - Track the state of the SCSI session, i.e. Command, Data or Response phase.
  - Allow the flow to be redirected in the Command Phase
  - Provide any additional support that may be required by SAN Virtualization software

SSL Support

- For SSL, the main requirement is to transfer complete SSL records across the interface to and from an external SSL device.
- The Interface Core will
  - peek at the content of incoming packets
  - Determine the Length of the SSL record from the SSL header
  - Only transfer complete SSL records across the interface
  - Support a Host programmable index for accelerating the SSL accelerator's table lookup.

Typical System



- Maps SPI-4 to external busses
- Manages Queues in CPU Memory

Astute Networks